

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

data = pd.read_csv("/content/Energy efficiencigy.csv")
```

```
data.head(5)
```

↗

	LMK_KEY	ADDRESS1	POSTCODE	UPR	CURRENT_ENERGY_RATING	POTENTIAL_ENERGY_RATING	CURRENT_ENER
0	247421689062019061120570969138651	225, Westfield	CM18 6AP	5699849568	D	B	
1	672074579042011083012190381997908	170, Carters Mead	CM17 9EU	4035079868	C	C	
2	1696733829922019021205502602208431	11, Brickcroft Hoppit	CM17 9FJ	3096272678	C	B	
3	1638944259742018061021015856880218	174, Long Ley	CM20 3NW	782458578	D	B	
4	406114810922009121612110180158541	102, The Hides	CM20 3QP	4807140768	C	C	

5 rows × 71 columns

```
data.columns
```

```
↗ Index(['LMK_KEY', 'ADDRESS1', 'POSTCODE', 'UPR', 'CURRENT_ENERGY_RATING',
        'POTENTIAL_ENERGY_RATING', 'CURRENT_ENERGY_EFFICIENCY',
        'POTENTIAL_ENERGY_EFFICIENCY', 'PROPERTY_TYPE', 'BUILT_FORM',
        'INSPECTION_DATE', 'LOCAL_AUTHORITY', 'CONSTITUENCY', 'LODGEMENT_DATE',
        'TRANSACTION_TYPE', 'ENVIRONMENT_IMPACT_CURRENT',
        'ENVIRONMENT_IMPACT_POTENTIAL', 'ENERGY_CONSUMPTION_CURRENT',
        'ENERGY_CONSUMPTION_POTENTIAL', 'CO2_EMISSIONS_CURRENT',
        'CO2_EMISS_CURR_PER_FLOOR_AREA', 'CO2_EMISSIONS_POTENTIAL',
        'LIGHTING_COST_CURRENT', 'LIGHTING_COST_POTENTIAL',
        'HEATING_COST_CURRENT', 'HEATING_COST_POTENTIAL',
        'HOT_WATER_COST_CURRENT', 'HOT_WATER_COST_POTENTIAL',
        'TOTAL_FLOOR_AREA', 'ENERGY_TARIFF', 'MAINS_GAS_FLAG',
        'MAIN_HEATING_CONTROLS', 'MULTI_GLAZE_PROPORTION', 'GLAZED_TYPE',
        'GLAZED_AREA', 'EXTENSION_COUNT', 'NUMBER_HABITABLE_ROOMS',
        'NUMBER_HEATED_ROOMS', 'LOW_ENERGY_LIGHTING', 'NUMBER_OPEN_FIREPLACES',
        'HOTWATER_DESCRIPTION', 'HOT_WATER_ENERGY_EFF', 'HOT_WATER_ENV_EFF',
        'FLOOR_DESCRIPTION', 'WINDOWS_DESCRIPTION', 'WINDOWS_ENERGY_EFF',
        'WINDOWS_ENV_EFF', 'WALLS_DESCRIPTION', 'WALLS_ENERGY_EFF',
        'WALLS_ENV_EFF', 'SECONDHEAT_DESCRIPTION', 'ROOF_DESCRIPTION',
        'ROOF_ENERGY_EFF', 'ROOF_ENV_EFF', 'MAINHEAT_DESCRIPTION',
        'MAINHEAT_ENERGY_EFF', 'MAINHEAT_ENV_EFF', 'MAINHEATCONT_DESCRIPTION',
        'MAINHEATC_ENERGY_EFF', 'MAINHEATC_ENV_EFF', 'LIGHTING_DESCRIPTION',
        'LIGHTING_ENERGY_EFF', 'LIGHTING_ENV_EFF', 'MAIN_FUEL',
        'MECHANICAL_VENTILATION', 'ADDRESS', 'POSTTOWN', 'Unnamed: 67',
        'LODGEMENT_DATETIME', 'TENURE', 'UPRN_SOURCE'],
        dtype='object')
```

we want to understnd how the data we interact with the model before preprocessing this is will prpvide a benchmark for preprocessing and how to improve the model

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

for column in data.columns:
    mode = data[column].mode()[0]
    data[column].fillna(mode, inplace=True)
```

 /tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)


```
data[column].fillna(mode, inplace=True)
```

/tmp/ipython-input-1864977839.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values is a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value, inplace=True)

```
data[column].fillna(mode, inplace=True)
```

```
data.isnull().sum()
```



	0
<b>LMK_KEY</b>	0
<b>ADDRESS1</b>	0
<b>POSTCODE</b>	0
<b>UPR</b>	0
<b>CURRENT_ENERGY_RATING</b>	0
...	...
<b>POSTTOWN</b>	0
<b>Unnamed: 67</b>	0
<b>LODGE_MENT_DATETIME</b>	0
<b>TENURE</b>	0
<b>UPRN_SOURCE</b>	0

71 rows × 1 columns

dtype: int64

data.columns

```
Index(['LMK_KEY', 'ADDRESS1', 'POSTCODE', 'UPR', 'CURRENT_ENERGY_RATING',
      'POTENTIAL_ENERGY_RATING', 'CURRENT_ENERGY_EFFICIENCY',
      'POTENTIAL_ENERGY_EFFICIENCY', 'PROPERTY_TYPE', 'BUILT_FORM',
      'INSPECTION_DATE', 'LOCAL_AUTHORITY', 'CONSTITUENCY', 'LODGEMENT_DATE',
      'TRANSACTION_TYPE', 'ENVIRONMENT_IMPACT_CURRENT',
      'ENVIRONMENT_IMPACT_POTENTIAL', 'ENERGY_CONSUMPTION_CURRENT',
      'ENERGY_CONSUMPTION_POTENTIAL', 'CO2_EMISSIONS_CURRENT',
      'CO2_EMISS_CURR_PER_FLOOR_AREA', 'CO2_EMISSIONS_POTENTIAL',
      'LIGHTING_COST_CURRENT', 'LIGHTING_COST_POTENTIAL',
      'HEATING_COST_CURRENT', 'HEATING_COST_POTENTIAL',
      'HOT_WATER_COST_CURRENT', 'HOT_WATER_COST_POTENTIAL',
      'TOTAL_FLOOR_AREA', 'ENERGY_TARIFF', 'MAINS_GAS_FLAG',
      'MAIN_HEATING_CONTROLS', 'MULTI_GLAZE_PROPORTION', 'GLAZED_TYPE',
      'GLAZED_AREA', 'EXTENSION_COUNT', 'NUMBER_HABITABLE_ROOMS',
      'NUMBER_HEATED_ROOMS', 'LOW_ENERGY_LIGHTING', 'NUMBER_OPEN_FIREPLACES',
      'HOTWATER_DESCRIPTION', 'HOT_WATER_ENERGY_EFF', 'HOT_WATER_ENV_EFF',
      'FLOOR_DESCRIPTION', 'WINDOWS_DESCRIPTION', 'WINDOWS_ENERGY_EFF',
      'WINDOWS_ENV_EFF', 'WALLS_DESCRIPTION', 'WALLS_ENERGY_EFF',
      'WALLS_ENV_EFF', 'SECONDHEAT_DESCRIPTION', 'ROOF_DESCRIPTION',
      'ROOF_ENERGY_EFF', 'ROOF_ENV_EFF', 'MAINHEAT_DESCRIPTION',
      'MAINHEAT_ENERGY_EFF', 'MAINHEAT_ENV_EFF', 'MAINHEATCONT_DESCRIPTION',
      'MAINHEATC_ENERGY_EFF', 'MAINHEATC_ENV_EFF', 'LIGHTING_DESCRIPTION',
      'LIGHTING_ENERGY_EFF', 'LIGHTING_ENV_EFF', 'MAIN_FUEL',
      'MECHANICAL_VENTILATION', 'ADDRESS', 'POSTTOWN', 'Unnamed: 67',
      'LODGEMENT_DATETIME', 'TENURE', 'UPRN_SOURCE'],
      dtype='object')
```

```
data = data.drop(columns=['LMK_KEY', 'ADDRESS1', 'LODGEMENT_DATE', 'LODGEMENT_DATETIME', 'INSPECTION_DATE'])
```

Double-click (or enter) to edit

```
ordinal_features =[
    "CURRENT_ENERGY_RATING",
    "POTENTIAL_ENERGY_RATING",
    "HOT_WATER_ENERGY_EFF",
    "HOT_WATER_ENV_EFF",
    "WINDOWS_ENERGY_EFF",
    "WINDOWS_ENV_EFF",
    "WALLS_ENERGY_EFF",
    "WALLS_ENV_EFF",
    "ROOF_ENERGY_EFF",
    "ROOF_ENV_EFF",
    "MAINHEAT_ENERGY_EFF",
    "MAINHEAT_ENV_EFF",
    "MAINHEATC_ENERGY_EFF",
    "MAINHEATC_ENV_EFF",
    "LIGHTING_ENERGY_EFF",
    "LIGHTING_ENV_EFF"
]
```

```
Categorical_features = data.select_dtypes(include=['object']).columns
```

```
data.head(2)
```

	POSTCODE	UPR	CURRENT_ENERGY_RATING	POTENTIAL_ENERGY_RATING	CURRENT_ENERGY_EFFICIENCY	POTENTIAL_ENERGY_EFFICIENCY	PRC
0	CM186AP	5699849568	D	B	60		84
1	CM179EU	4035079868	C	C	69		70

2 rows × 66 columns

Categorical\_features

```
Index(['POSTCODE', 'CURRENT_ENERGY_RATING', 'POTENTIAL_ENERGY_RATING',
      'PROPERTY_TYPE', 'BUILT_FORM', 'LOCAL_AUTHORITY', 'CONSTITUENCY',
      'TRANSACTION_TYPE', 'ENERGY_TARIFF', 'MAINS_GAS_FLAG', 'GLAZED_TYPE',
      'GLAZED_AREA', 'HOTWATER_DESCRIPTION', 'HOT_WATER_ENERGY_EFF',
      'HOT_WATER_ENV_EFF', 'FLOOR_DESCRIPTION', 'WINDOWS_DESCRIPTION',
      'WINDOWS_ENERGY_EFF', 'WINDOWS_ENV_EFF', 'WALLS_DESCRIPTION',
      'WALLS_ENERGY_EFF', 'WALLS_ENV_EFF', 'SECONDHEAT_DESCRIPTION',
      'ROOF_DESCRIPTION', 'ROOF_ENERGY_EFF', 'ROOF_ENV_EFF',
      'MAINHEAT_DESCRIPTION', 'MAINHEAT_ENERGY_EFF', 'MAINHEAT_ENV_EFF',
      'MAINHEATCONT_DESCRIPTION', 'MAINHEATC_ENERGY_EFF', 'MAINHEATC_ENV_EFF',
```

```

'LIGHTING_DESCRIPTION', 'LIGHTING_ENERGY_EFF', 'LIGHTING_ENV_EFF',
'MAIN_FUEL', 'MECHANICAL_VENTILATION', 'ADDRESS', 'POSTTOWN',
'Unnamed: 67', 'TENURE', 'UPRN_SOURCE'],
dtype='object')

```

```

# Define the category lists

```

```

energy_rating_categories = ["G", "F", "E", "D", "C", "B", "A"]

```

```

efficiency_categories = ["Very Poor", "Poor", "Average", "Good", "Very Good", "Excellent"]

```

```

# Create the list of categories for all 16 features

```

```

categories = [energy_rating_categories, energy_rating_categories] + [efficiency_categories] * 14

```

```

# Initialize and fit the OrdinalEncoder

```

```

order_encoder = OrdinalEncoder(categories=categories)

```

```

data[ordinal_features] = order_encoder.fit_transform(data[ordinal_features])

```

```

label_encoder = LabelEncoder()

```

```

for column in Categorical_features:

```

```

    data[column] = label_encoder.fit_transform(data[column])

```

```

data

```



	POSTCODE	UPR	CURRENT_ENERGY_RATING	POTENTIAL_ENERGY_RATING	CURRENT_ENERGY_EFFICIENCY	POTENTIAL_ENERGY_EFFICIENCY
0	384	5699849568	3	5	60	8
1	223	4035079868	4	4	69	7
2	233	3096272678	4	5	72	8
3	1332	782458578	3	5	66	8
4	1362	4807140768	4	4	72	7
...	...	...	...	...	...	...
47461	731	10007592502	3	5	64	8
47462	474	4004917378	4	4	76	7
47463	364	10007542914	5	5	83	8
47464	918	10007604622	4	5	70	8
47465	1370	10007663254	4	4	72	7

47466 rows × 66 columns

```

x = data.drop(columns=['CURRENT_ENERGY_EFFICIENCY'])

```

```

y = data['CURRENT_ENERGY_EFFICIENCY']

```

```

from sklearn.model_selection import train_test_split

```

```

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

```

```

model=RandomForestRegressor()

```

```

model.fit(x_train, y_train)

```

```

r_squared = model.score(x_test, y_test)

```

```

print(f"R-squared: {r_squared}")

```



R-squared: 0.9918313243693793

```

from sklearn.metrics import mean_squared_error

```

```

y_pred = model.predict(x_test)

```

```

mse = mean_squared_error(y_test, y_pred)

```

```

print(f"Mean Squared Error: {mse}")

```



Mean Squared Error: 0.76223795028439

```

plt.figure(figsize=(10, 6))

```

```

plt.scatter(y_test, y_pred)

```

```

plt.xlabel("Actual Values")

```

```

plt.ylabel("Predicted Values")

```

```

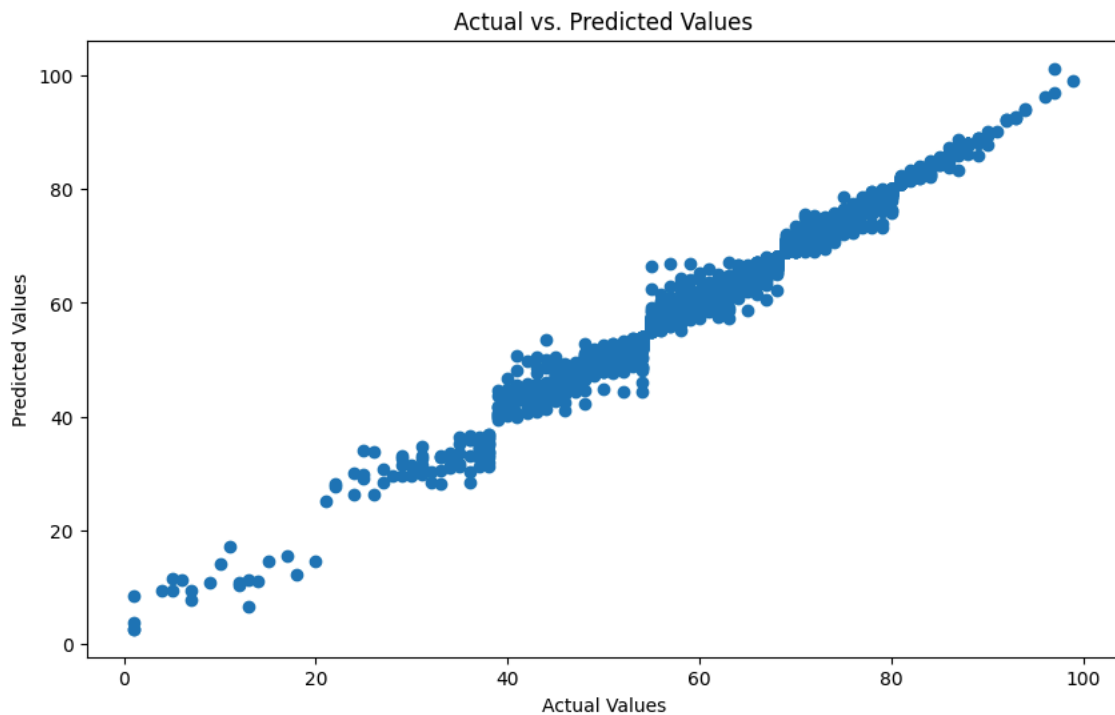
plt.title("Actual vs. Predicted Values")

```

```

plt.show()

```



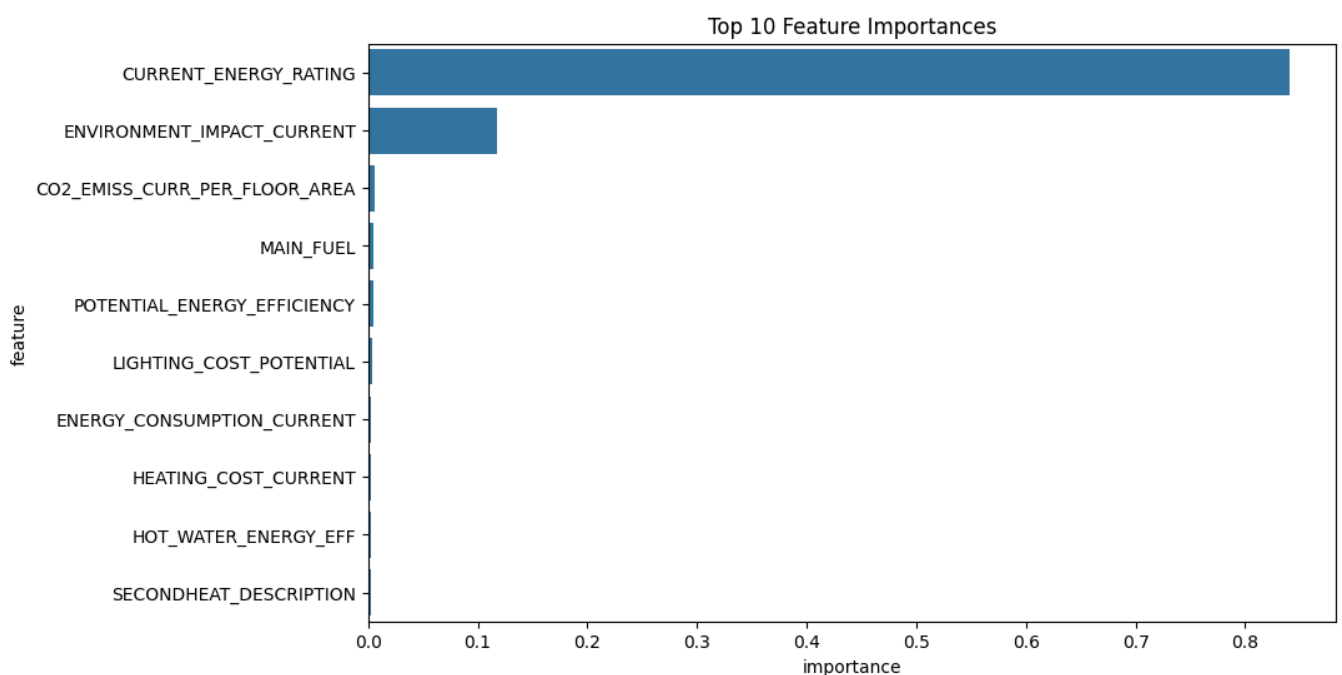
- we want to understand which features are most significant, because it would be tiring to input over 20 features into a
- ✓ Model to make prediction. we are going to use features importance to determine the most dominant features in the model .

```
# Get feature importances from the trained model
importances = model.feature_importances_

# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})

# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



- These model is dominated by Current\_Energy\_rating which i feel is making the model bias, we are going to drop these features and see how the model behave.

```
x = data.drop(columns=['CURRENT_ENERGY_EFFICIENCY', 'CURRENT_ENERGY_RATING', 'ENVIRONMENT_IMPACT_CURRENT'])
y = data['CURRENT_ENERGY_EFFICIENCY']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model=RandomForestRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

R-squared: 0.9845827077065052

```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

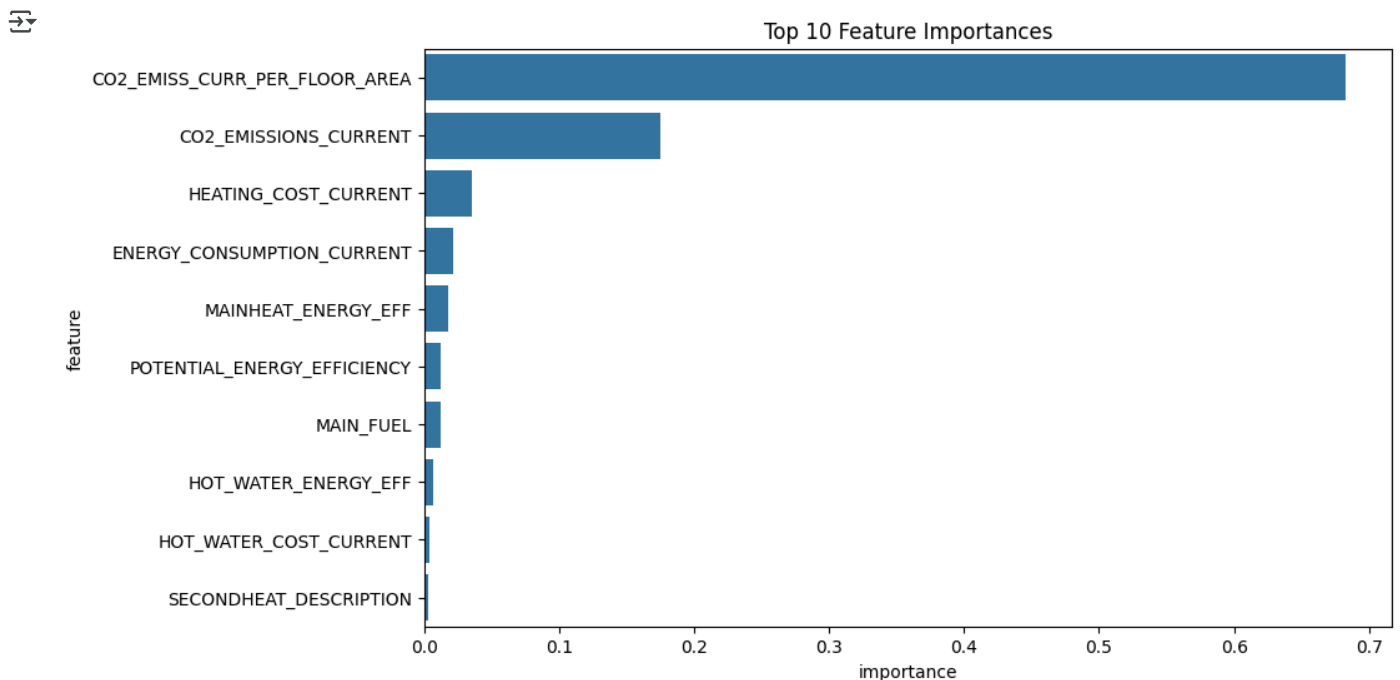
Mean Squared Error: 1.4386230777333056

```
# Get feature importances from the trained model
importances = model.feature_importances_
```

```
# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})
```

```
# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)
```

```
# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



- Co2 emmisssion seems to be dominating the model , but in reality CO2 Emmison might not be a handy information to go by we are going to reduce the features further to see how the model perform

```
x = x.drop(columns=['CO2_EMISSIONS_CURRENT', 'CO2_EMISS_CURR_PER_FLOOR_AREA', 'CO2_EMISSIONS_POTENTIAL'])
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model=RandomForestRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

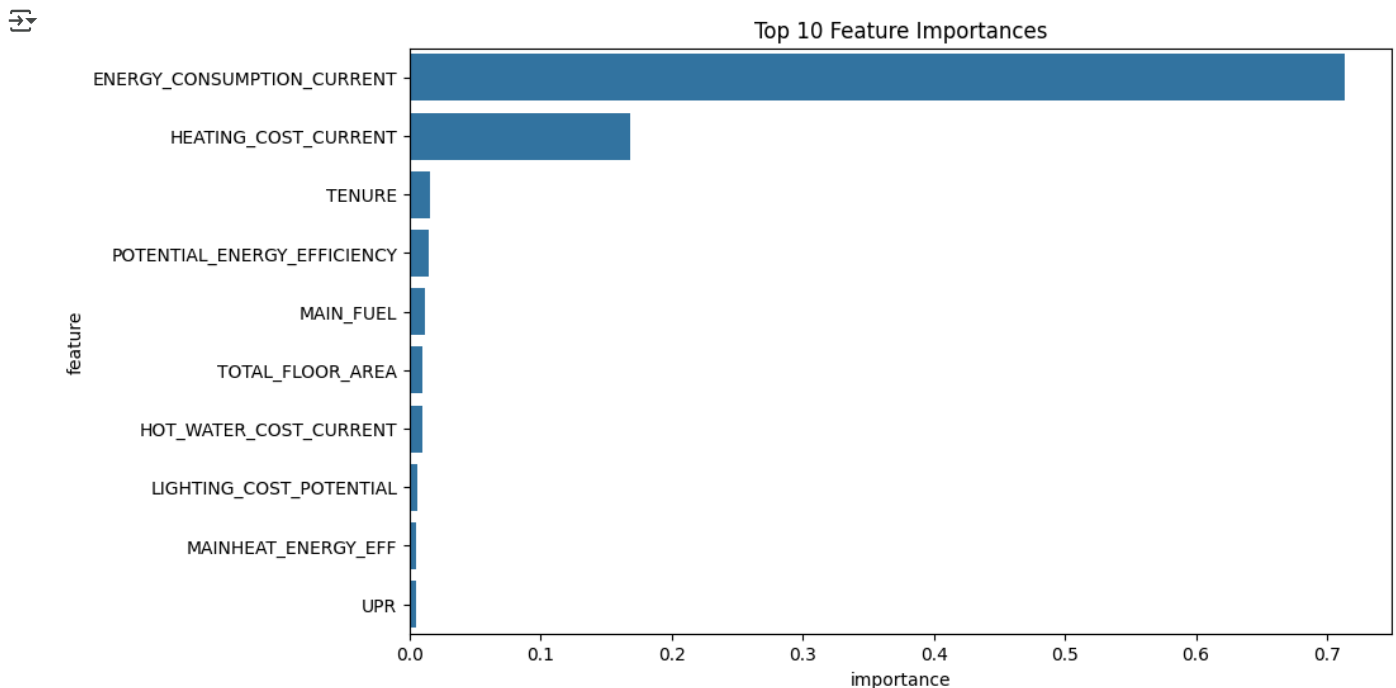
↗ R-squared: 0.979321655039248

```
importances = model.feature_importances_

# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})

# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



▼

While Energy Consumption and Heating cost were highly predictive, they are derived from the same energy assessments that generate the target variable.

Including them would introduce **data leakage**, inflating performance but making the model unusable in real-world scenarios where such data is unavailable prior to inspection.

These features will be removed, and the model retrained using only realistic features.

```
x.drop(columns=['HOT_WATER_COST_CURRENT', 'ENERGY_CONSUMPTION_POTENTIAL', 'POTENTIAL_ENERGY_EFFICIENCY', 'HOT_WATER_COST_POTENTIAL', 'HEATING_COST_CURRENT', 'ENERGY_CONSUMPTION_CURRENT'])
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model=RandomForestRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

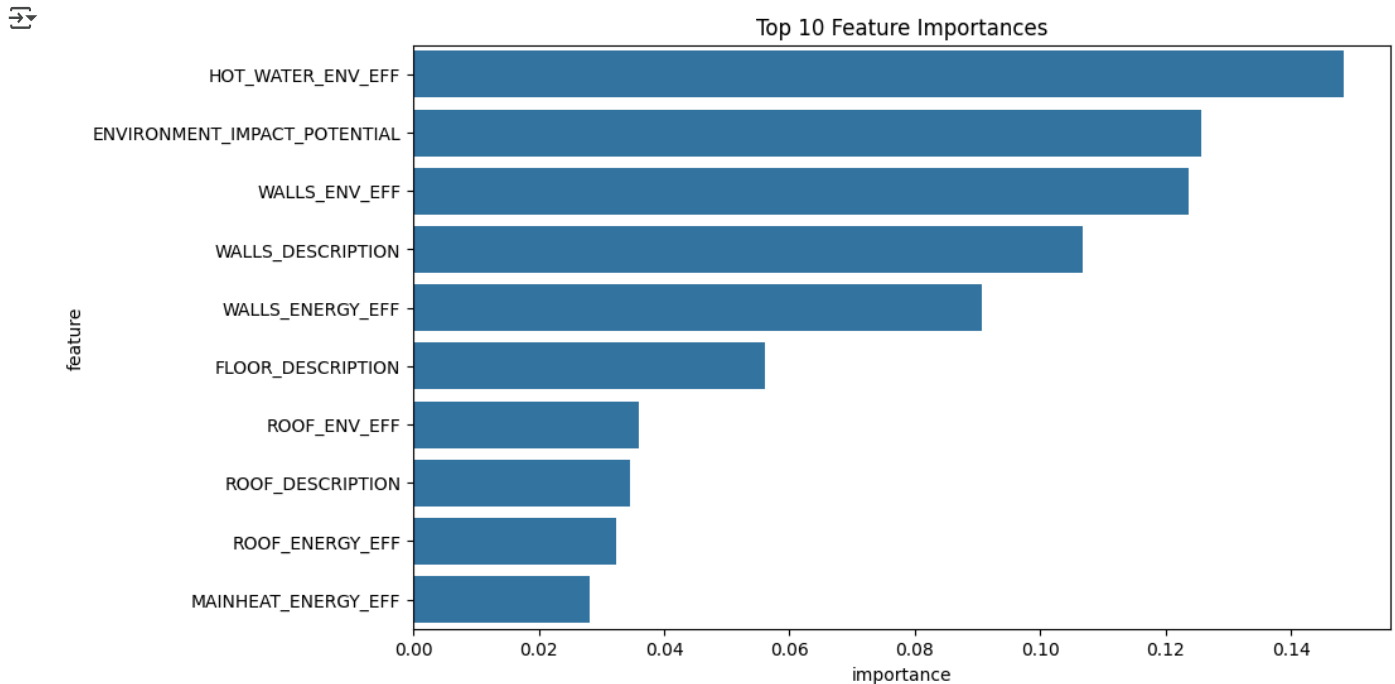
↗ R-squared: 0.8865204021318152

```
importances = model.feature_importances_

# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})
```

```
# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



```
features_to_drop = [
    'HOT_WATER_ENV_EFF',
    'ENVIRONMENT_IMPACT_POTENTIAL',
    'WALLS_ENERGY_EFF',
    'WALLS_ENV_EFF',
    'ROOF_ENV_EFF',
    'ROOF_ENERGY_EFF',
    'MAINHEAT_ENERGY_EFF', 'MAINHEAT_ENV_EFF', 'MAINHEAT_DESCRIPTION', 'LIGHTING_DESCRIPTION',
    'MAINHEATC_ENERGY_EFF', 'LOW_ENERGY_LIGHTING',
    'MAINHEATC_ENV_EFF', 'Unnamed: 67', 'UPR', 'MAIN_HEATING_CONTROLS', 'MAINHEATCONT_DESCRIPTION', 'LIGHTING_ENERGY_EFF', 'LIGHTING_ENV_EFF'
]
```

```
x = x.drop(columns=features_to_drop)
```

```
x.head(2)
```

	POSTCODE	PROPERTY_TYPE	BUILT_FORM	LOCAL_AUTHORITY	CONSTITUENCY	TRANSACTION_TYPE	TOTAL_FLOOR_AREA	MULTI_GLAZE_PROPORTION	G
0	384	2	3	0	0	7	138.0	100.0	
1	223	3	5	0	0	16	85.0	100.0	

2 rows × 27 columns

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model=RandomForestRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

```
R-squared: 0.7235389049290717
```

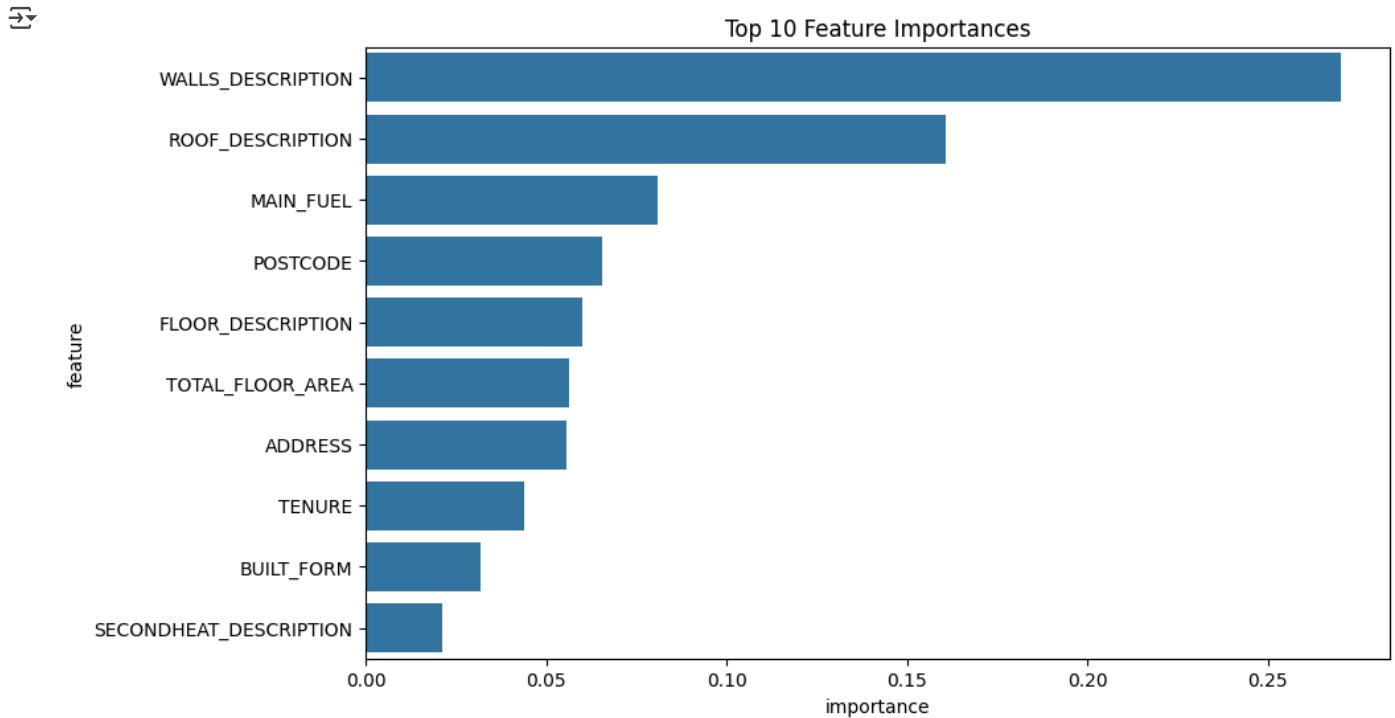
```
importances = model.feature_importances_
```

```
# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})
```

```
# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)
```



```
# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

Mean Squared Error: 25.797221969533346

## ✓ Top Feature Selection for Model Testing

We selected the top 10 most important features from the original model to evaluate performance with a reduced and interpretable set. The feature ADDRESS was excluded even though it ranked high in importance. This is because:

- It poses a **risk of overfitting** due to its specificity.
- It may introduce **bias or redundancy**, especially when combined with POSTCODE .
- It lacks **generalization value** for new/unseen addresses.

The remaining top features include key attributes such as wall and roof descriptions, main heating controls, and total floor area.

```
x = x[["WALLS_DESCRIPTION", "POSTCODE", "ROOF_DESCRIPTION", "FLOOR_DESCRIPTION", "TOTAL_FLOOR_AREA", "MAIN_FUEL", "NUMBER_HEATED_ROOMS", "NUMB
```

```
x['avg_room_area'] = x['TOTAL_FLOOR_AREA']/x['NUMBER_HABITABLE_ROOMS']
x["Avg_heated_area"] = x["avg_room_area"]*x["NUMBER_HEATED_ROOMS"]
```

x

	WALLS_DESCRIPTION	POSTCODE	ROOF_DESCRIPTION	FLOOR_DESCRIPTION	TOTAL_FLOOR_AREA	MAIN_FUEL	NUMBER_HEATED_ROOMS	NUMBER_HA
0		98	384	68	70	138.00	21	6.0
1		119	223	72	1	85.00	21	4.0
2		97	233	59	68	81.00	15	3.0
3		119	1332	74	66	71.00	21	3.0
4		119	1362	0	66	46.43	22	2.0
...	...	...	...	...	...	...	...	...
47461		114	731	99	0	59.00	21	3.0
47462		97	474	0	64	43.00	21	3.0
47463		8	364	1	5	48.00	1	3.0
47464		100	918	97	66	96.00	21	5.0
47465		99	1370	0	66	46.00	21	2.0

47466 rows × 10 columns

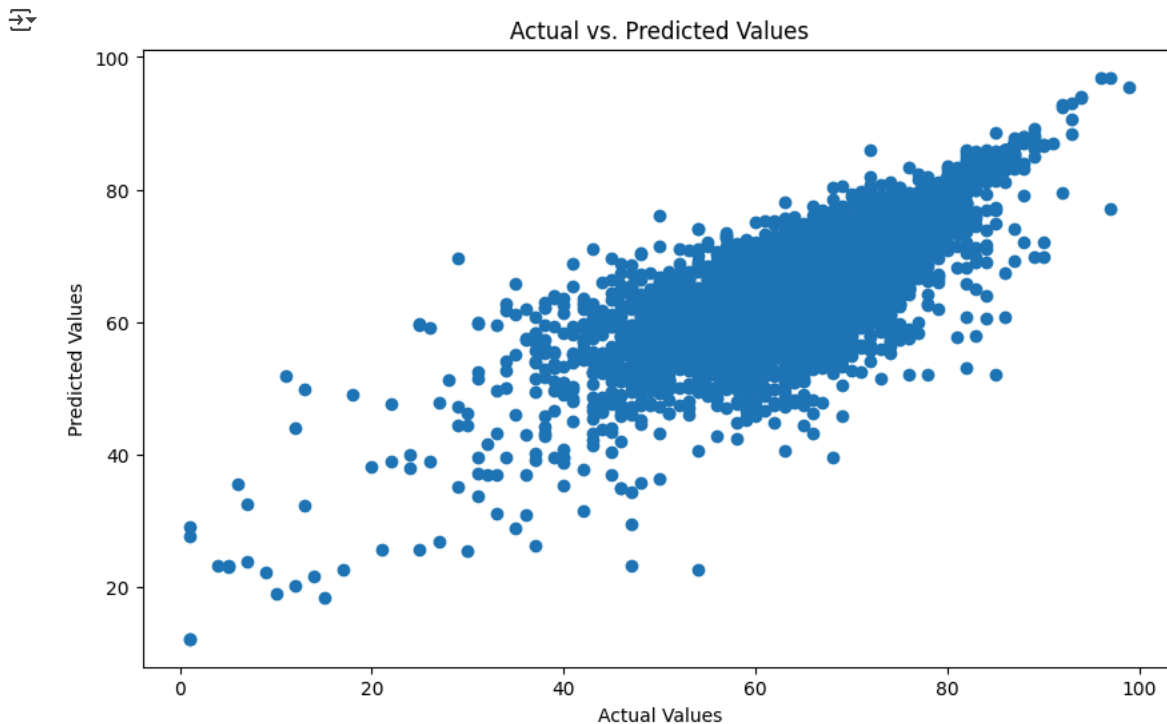
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model=RandomForestRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

R-squared: 0.6443267949107855

```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

Mean Squared Error: 33.18868652367816

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```



- ✓ This means your model explains about 64% of the variance in the actual values, and from the scatter plot we can see that the spread follows an upward spread but its quite unnoticeable. we are now going to see how e can improve this model.

↻

	WALLS_DESCRIPTION	POSTCODE	ROOF_DESCRIPTION	FLOOR_DESCRIPTION	TOTAL_FLOOR_AREA	MAIN_FUEL	NUMBER_HEATED_ROOMS	NUMBER_HA
0	98	384	68	70	138.00	21	6.0	
1	119	223	72	1	85.00	21	4.0	
2	97	233	59	68	81.00	15	3.0	
3	119	1332	74	66	71.00	21	3.0	
4	119	1362	0	66	46.43	22	2.0	
...	...	...	...	...	...	...	...	...
47461	114	731	99	0	59.00	21	3.0	
47462	97	474	0	64	43.00	21	3.0	
47463	8	364	1	5	48.00	1	3.0	
47464	100	918	97	66	96.00	21	5.0	
47465	99	1370	0	66	46.00	21	2.0	

47466 rows × 10 columns

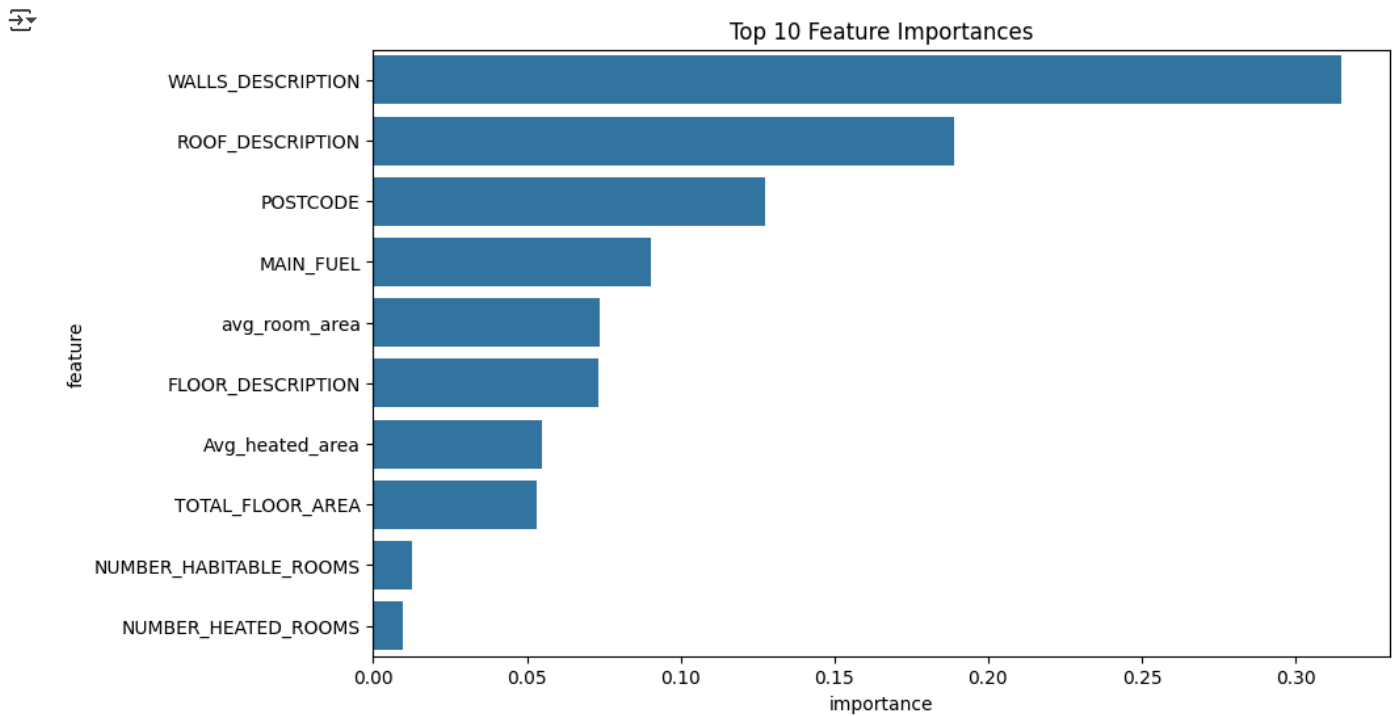
Start coding or [generate](#) with AI.

```
importances = model.feature_importances_

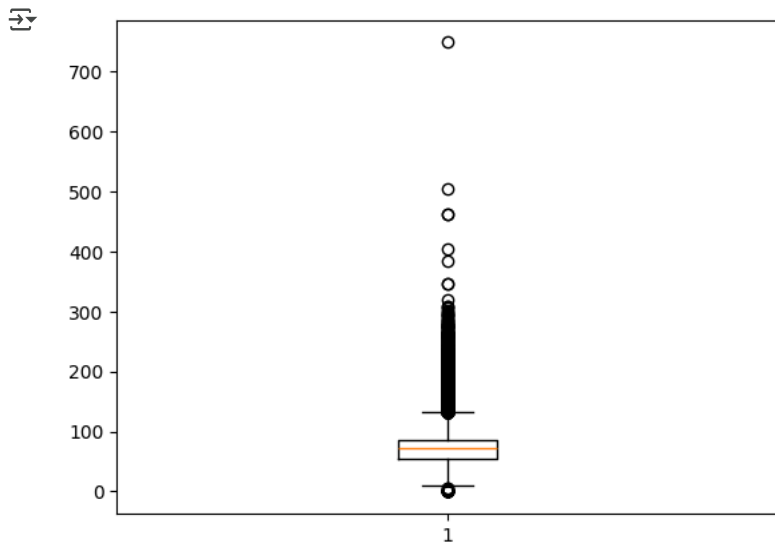
# Create a dataframe of feature importances
feature_importances = pd.DataFrame({'feature': x.columns, 'importance': importances})

# Sort the features by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)

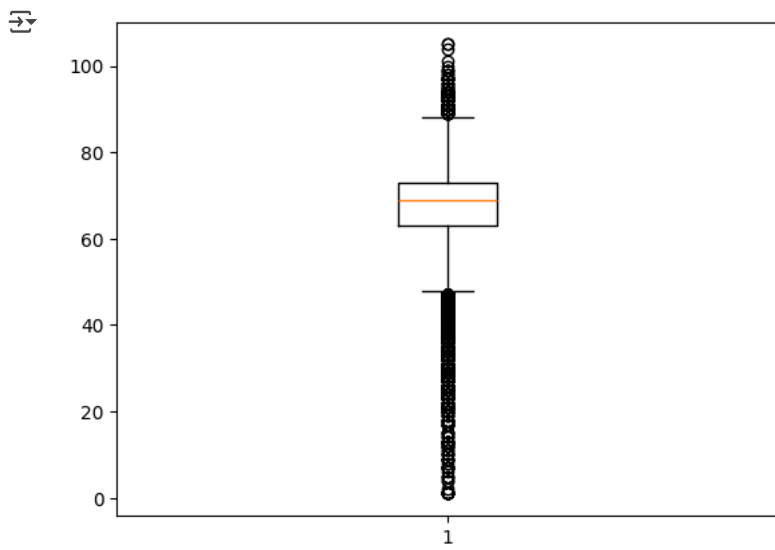
# Plot the feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importances.head(10))
plt.title('Top 10 Feature Importances')
plt.show()
```



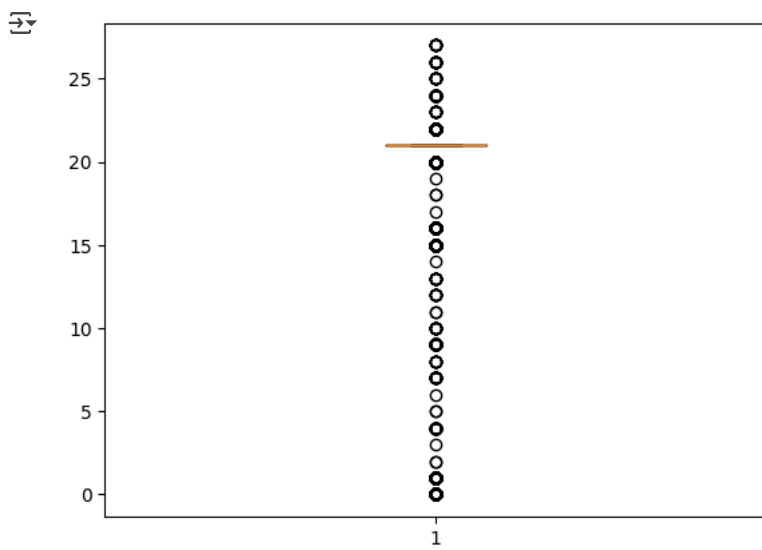
```
plt.boxplot(x=x['TOTAL_FLOOR_AREA'])
plt.show()
```



```
plt.boxplot(x=y)
plt.show()
```

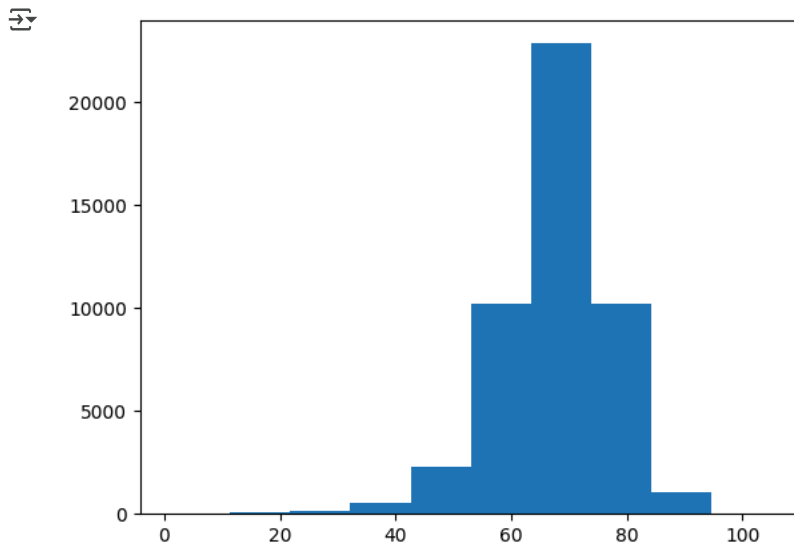


```
plt.boxplot(x=x['MAIN_FUEL'])
plt.show()
```



```
import numpy as np
```

```
plt.hist(np.array(y))
plt.show()
```



```
from sklearn.ensemble import GradientBoostingRegressor
model= GradientBoostingRegressor()
model.fit(x_train, y_train)
r_squared = model.score(x_test, y_test)
print(f"R-squared: {r_squared}")
```

R-squared: 0.6198765735253268

```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

Mean Squared Error: 35.470193034108085

x

	WALLS_DESCRIPTION	POSTCODE	ROOF_DESCRIPTION	FLOOR_DESCRIPTION	TOTAL_FLOOR_AREA	MAIN_FUEL	NUMBER_HEATED_ROOMS	NUMBER_HA
0	98	384	68	70	138.00	21	6.0	
1	119	223	72	1	85.00	21	4.0	
2	97	233	59	68	81.00	15	3.0	
3	119	1332	74	66	71.00	21	3.0	
4	119	1362	0	66	46.43	22	2.0	
...	...	...	...	...	...	...	...	...
47461	114	731	99	0	59.00	21	3.0	
47462	97	474	0	64	43.00	21	3.0	
47463	8	364	1	5	48.00	1	3.0	
47464	100	918	97	66	96.00	21	5.0	
47465	99	1370	0	66	46.00	21	2.0	

47466 rows × 10 columns

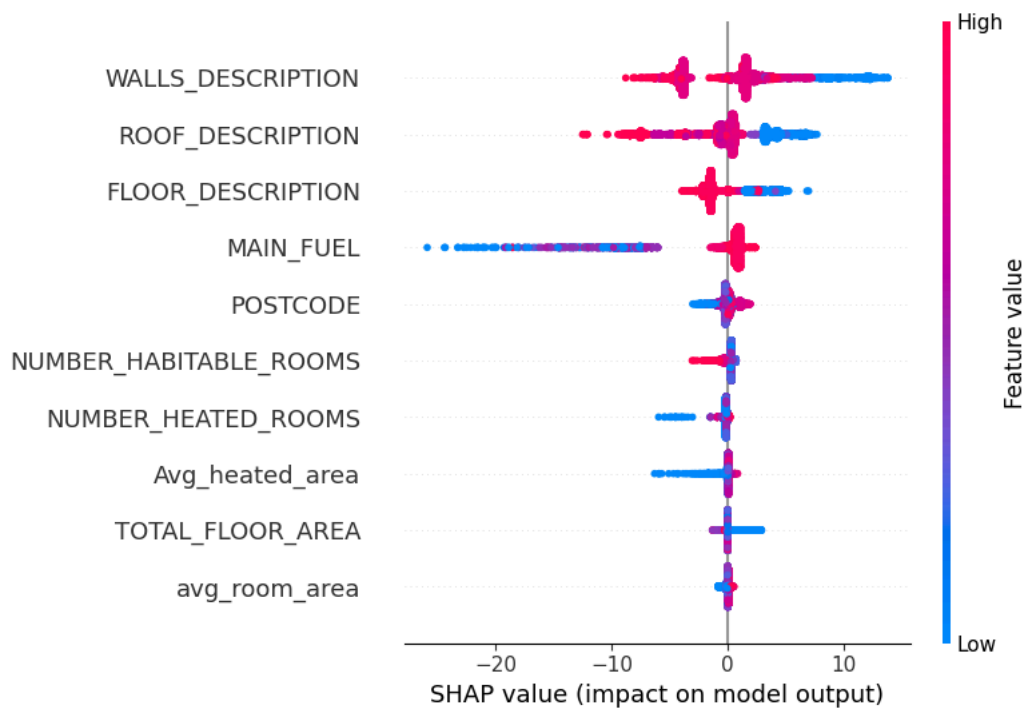
```
import shap
```

```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test)
```

```
shap.dependence_plot("POSTCODE", shap_values, x_test)
```

```
shap.force_plot(explainer.expected_value, shap_values[0,:], x_test.iloc[0,:])
```



2 |

14.00