databricks IOTH731_EnergyEfficiency_Project

(https://databricks.com)

Reading and Understanding Data

- X1 Relative Compactness

- X2 Surface Area

- X3 Wall Area

- X4 Roof Area

- X5 Overall Height

- X6 Orientation

- X7 Glazing Area

- X8 Glazing Area Distribution

- y1 Heating Load

- y2 Cooling Load

```
# File location and type
file_location = "/FileStore/tables/ENB2012_data-1.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
dbf = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(dbf)
```

Table

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
1	0.98	514.50	294.00	110.25	7.00	2	0.00	0	15.55	21.33
2	0.98	514.50	294.00	110.25	7.00	3	0.00	0	15.55	21.33
3	0.98	514.50	294.00	110.25	7.00	4	0.00	0	15.55	21.33
4	0.98	514.50	294.00	110.25	7.00	5	0.00	0	15.55	21.33
5	0.90	563.50	318.50	122.50	7.00	2	0.00	0	20.84	28.28
6	0.90	563.50	318.50	122.50	7.00	3	0.00	0	21.46	25.38
7	0.90	563.50	318.50	122.50	7.00	4	0.00	0	20.71	25.16
8	0.90	563.50	318.50	122.50	7.00	5	0.00	0	19.68	29.60
9	0.86	588.00	294.00	147.00	7.00	2	0.00	0	19.50	27.30
10	0.86	588.00	294.00	147.00	7.00	3	0.00	0	19.95	21.97
11	0.86	588.00	294.00	147.00	7.00	4	0.00	0	19.34	23.49
12	0.86	588.00	294.00	147.00	7.00	5	0.00	0	18.31	27.87
13	0.82	612.50	318.50	147.00	7.00	2	0.00	0	17.05	23.77
14	0.82	612.50	318.50	147.00	7.00	3	0.00	0	17.41	21.46

768 rows

```
# import required libraries for clustering
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

1 Convert Data from Spark to Pandas DataFrame

2 Rename column names to meaningfull names

3 Convert column data types to original datatypes

```
# Convert Spark DataFrame to Pandas DataFrame
df = dbf.toPandas()

#Rename the columns
df.rename(columns={'X1':'Relative_Compactness','X2':'Surface_Area',
                   'X3':'Wall_Area','X4':'Roof_Area','X5':'Overall_Height',
                   'X6':'Orientation','X7':'Glazing_Area',
                   'X8':'Glazing_Area_Distribution','Y1':'Heating_Load',
                   'Y2':'Cooling_Load'}, inplace=True)

df
```

	Relative_Compactness	Surface_Area	Wall_Area	Roof_Area	Overall_Height	Orientation	Glazing_Area	Glazing_Area_Distribution	Heating_Load
0	0.98	514.50	294.00	110.25	7.00	2	0.00	0	15.55
1	0.98	514.50	294.00	110.25	7.00	3	0.00	0	15.55
2	0.98	514.50	294.00	110.25	7.00	4	0.00	0	15.55
3	0.98	514.50	294.00	110.25	7.00	5	0.00	0	15.55
4	0.90	563.50	318.50	122.50	7.00	2	0.00	0	20.84
...	...	...	...	...	...	...	...	...	...
763	0.64	784.00	343.00	220.50	3.50	5	0.40	5	17.88
764	0.62	808.50	367.50	220.50	3.50	2	0.40	5	16.54
765	0.62	808.50	367.50	220.50	3.50	3	0.40	5	16.44
766	0.62	808.50	367.50	220.50	3.50	4	0.40	5	16.48
767	0.62	808.50	367.50	220.50	3.50	5	0.40	5	16.64

768 rows × 10 columns

```
# Convert data types of specific columns
df = df.astype({
    'Relative_Compactness': 'float64',
    'Surface_Area': 'float64',
    'Wall_Area': 'float64',
    'Roof_Area': 'float64',
    'Overall_Height': 'float64',
    'Orientation': 'int64',
    'Glazing_Area': 'float64',
    'Glazing_Area_Distribution': 'int64',
    'Heating_Load': 'float64',
    'Cooling_Load': 'float64'
})

# Verify the changes
print(df.dtypes)
```

```
Relative_Compactness         float64
Surface_Area                 float64
Wall_Area                    float64
Roof_Area                    float64
Overall_Height               float64
Orientation                    int64
Glazing_Area                 float64
Glazing_Area_Distribution      int64
Heating_Load                 float64
Cooling_Load                 float64
dtype: object
```

Data Exploring

```
print(df.shape)
```

```
(768, 10)
```

```
print(df.isnull().sum())
```

```
Relative_Compactness         0
Surface_Area                 0
Wall_Area                    0
Roof_Area                    0
Overall_Height               0
Orientation                  0
Glazing_Area                 0
Glazing_Area_Distribution    0
Heating_Load                 0
Cooling_Load                 0
dtype: int64
```

```
df.describe([0, 0.05, 0.50, 0.95, 0.99, 1]).T
```

	count	mean	std	min	0%	5%	50%	95%	99%	100%	max
Relative_Compactness	768.0	0.764167	0.105777	0.62	0.62	0.6200	0.75	0.980	0.9800	0.98	0.98
Surface_Area	768.0	671.708333	88.086116	514.50	514.50	514.5000	673.75	808.500	808.5000	808.50	808.50
Wall_Area	768.0	318.500000	43.626481	245.00	245.00	245.0000	318.50	416.500	416.5000	416.50	416.50
Roof_Area	768.0	176.604167	45.165950	110.25	110.25	110.2500	183.75	220.500	220.5000	220.50	220.50
Overall_Height	768.0	5.250000	1.751140	3.50	3.50	3.5000	5.25	7.000	7.0000	7.00	7.00
Orientation	768.0	3.500000	1.118763	2.00	2.00	2.0000	3.50	5.000	5.0000	5.00	5.00
Glazing_Area	768.0	0.234375	0.133221	0.00	0.00	0.0000	0.25	0.400	0.4000	0.40	0.40
Glazing_Area_Distribution	768.0	2.812500	1.550960	0.00	0.00	0.0000	3.00	5.000	5.0000	5.00	5.00
Heating_Load	768.0	22.307201	10.090196	6.01	6.01	10.4635	18.95	39.860	42.0899	43.10	43.10
Cooling_Load	768.0	24.587760	9.513306	10.90	10.90	13.6175	22.08	40.037	45.5431	48.03	48.03

```
# categorise based on column types
def grab_col_names(dataframe, cat_th=15, car_th=20):
    #Catgeorical Variable Selection
    cat_cols = [col for col in dataframe.columns if str(dataframe[col].dtypes) in ["category","object","bool"]]
    num_but_cat = [col for col in dataframe.columns if dataframe[col].nunique() < cat_th and dataframe[col].dtypes in ["uint8","int
    cat_but_car = [col for col in dataframe.columns if dataframe[col].nunique() > car_th and str(dataframe[col].dtypes) in ["categ
    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    #Numerical Variable Selection
    num_cols = [col for col in dataframe.columns if dataframe[col].dtypes in ["uint8","int64","float64"]]
    num_cols = [col for col in num_cols if col not in cat_cols]

    return cat_cols, num_cols, cat_but_car, num_but_cat

cat_cols, num_cols, cat_but_car, num_but_cat = grab_col_names(df)
```

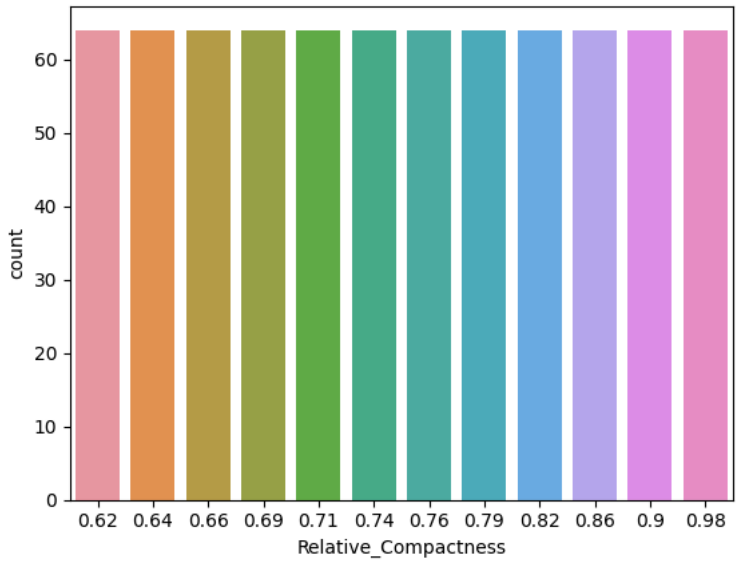
```
def cat_summary(dataframe,col_name,plot=False):
    print(pd.DataFrame({col_name: dataframe[col_name].value_counts(),
                        'Ration': 100 * dataframe[col_name].value_counts() / len(dataframe)}))
    if plot:
        sns.countplot(x=dataframe[col_name],data=dataframe)
        plt.show(block=True)
```

```
def cat_summary_df(dataframe):
    cat_cols, num_cols, cat_but_car, num_but_cat = grab_col_names(df)
    for col in cat_cols:
        cat_summary(dataframe, col, plot=True)
```

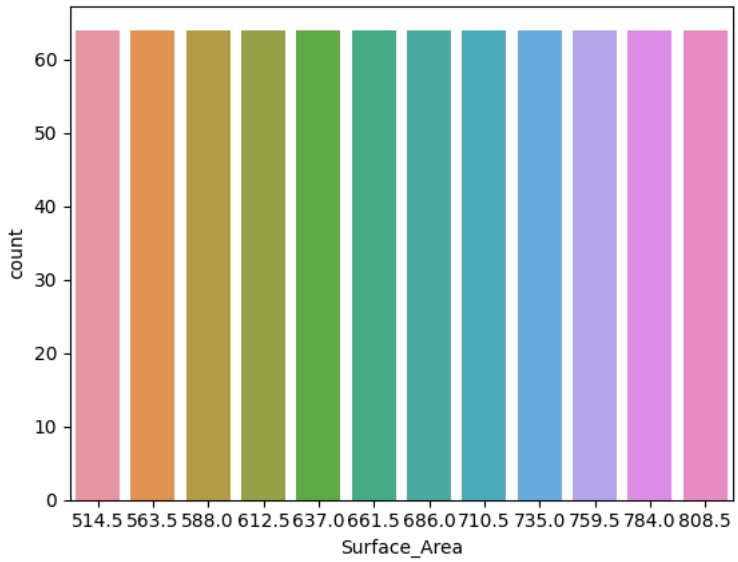
```
cat_summary_df(df)
```

```
     Relative_Compactness    Ration
0.98                   64  8.333333
0.90                   64  8.333333
0.86                   64  8.333333
0.82                   64  8.333333
0.79                   64  8.333333
0.76                   64  8.333333
0.74                   64  8.333333
0.71                   64  8.333333
0.69                   64  8.333333
0.66                   64  8.333333
```

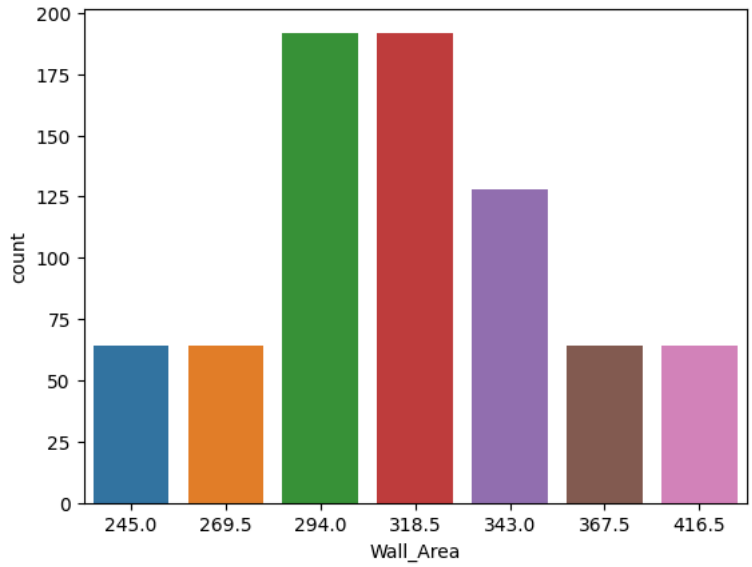
```
0.64                 64  8.333333
0.62                 64  8.333333
```



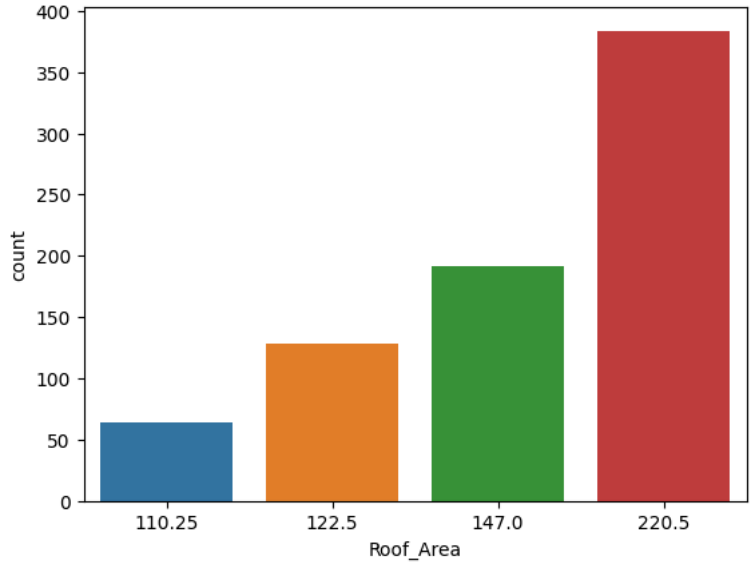

```
       Surface_Area    Ration
514.5            64  8.333333
563.5            64  8.333333
588.0            64  8.333333
612.5            64  8.333333
637.0            64  8.333333
661.5            64  8.333333
686.0            64  8.333333
710.5            64  8.333333
735.0            64  8.333333
759.5            64  8.333333
784.0            64  8.333333
808.5            64  8.333333
```



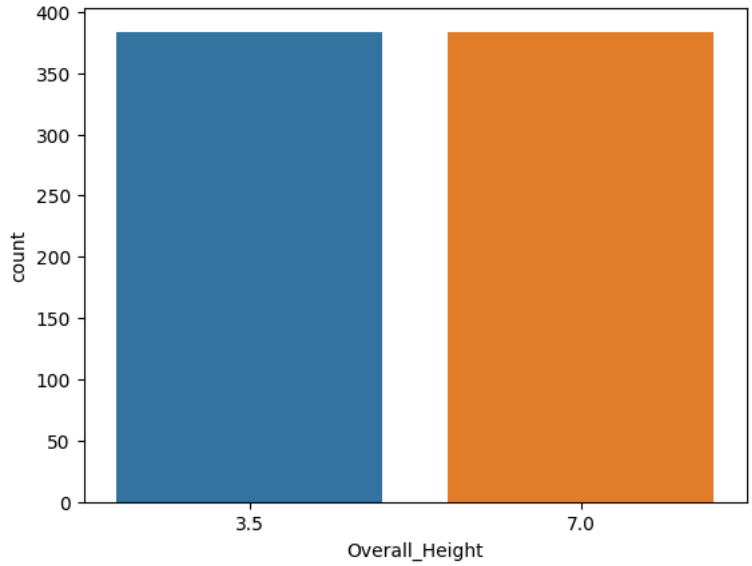

```
       Wall_Area     Ration
294.0        192  25.000000
318.5        192  25.000000
343.0        128  16.666667
416.5         64   8.333333
245.0         64   8.333333
269.5         64   8.333333
367.5         64   8.333333
```



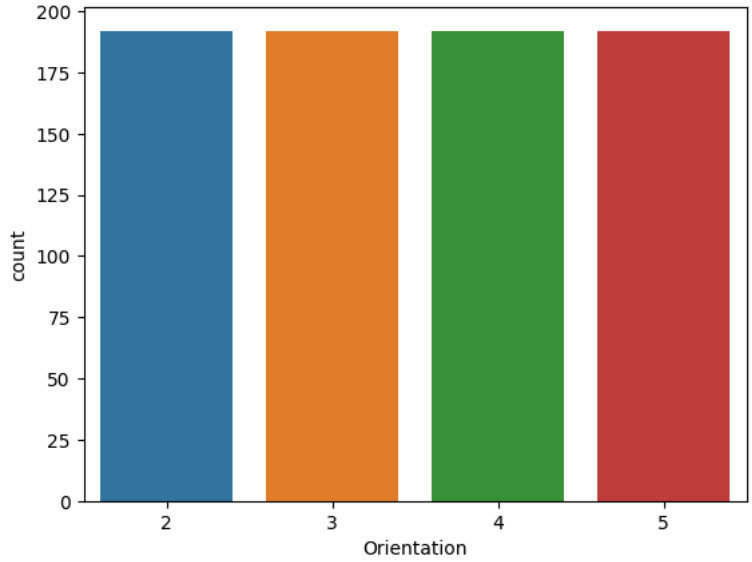
	Roof_Area	Ration
220.50	384	50.000000
147.00	192	25.000000
122.50	128	16.666667
110.25	64	8.333333



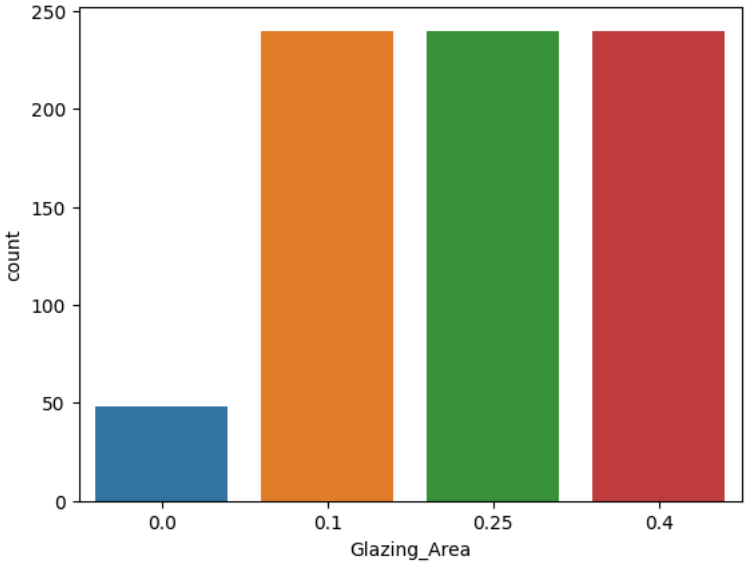
	Overall_Height	Ration
7.0	384	50.0
3.5	384	50.0



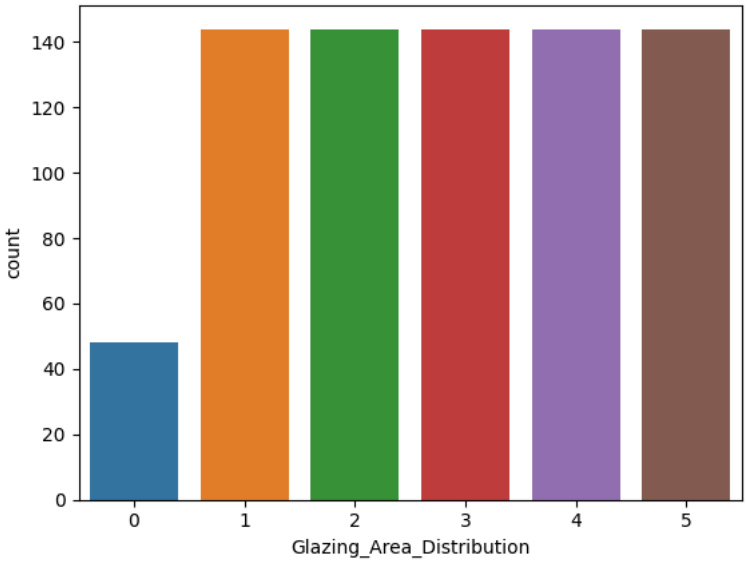
	Orientation	Ration
2	192	25.0
3	192	25.0
4	192	25.0
5	192	25.0



	Glazing_Area	Ration
0.10	240	31.25
0.25	240	31.25
0.40	240	31.25
0.00	48	6.25



	Glazing_Area_Distribution	Ration
1	144	18.75
2	144	18.75
3	144	18.75
4	144	18.75
5	144	18.75
0	48	6.25

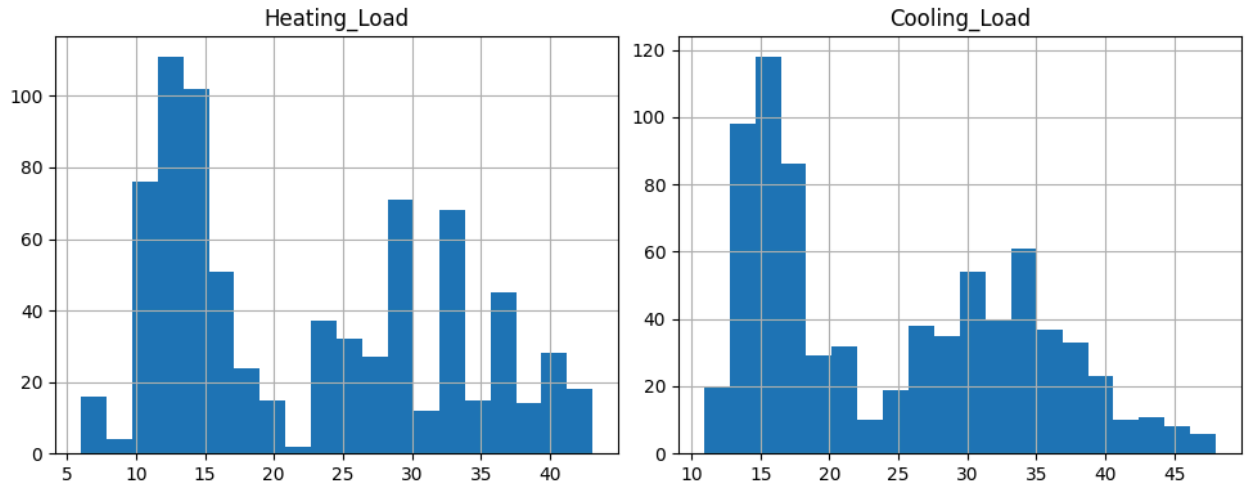


Analysis of Categorical and Numerical Variables

```
def plot_num_summary(dataframe):
    cat_cols, num_cols, cat_but_car, num_but_cat = grab_col_names(dataframe)
    plt.figure(figsize=(10,4))
    for index, col in enumerate(num_cols):
        plt.subplot(1,2,index+1)
        plt.tight_layout()
        dataframe[col].hist(bins=20)
        plt.title(col)

plot_num_summary(df)
```

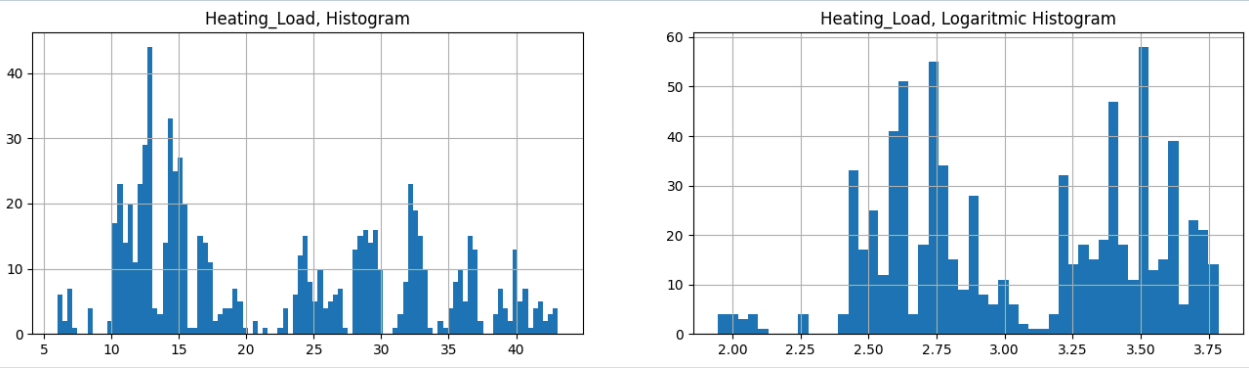
```
/root/.ipykernel/5280/command-3178850420648-91922312:6: UserWarning: The figure layout has changed to tight
  plt.tight_layout()
```



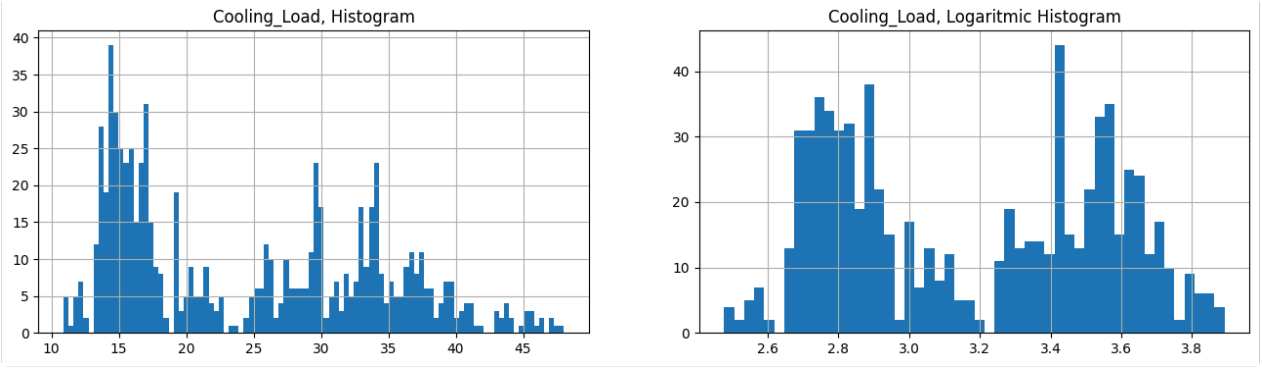
Examination of the dependent variable

```
def exam_dependent_variable(dataframe, target):
    plt.figure(figsize=(16,4))
    plt.subplot(121)
    dataframe[target].hist(bins=100)
    plt.title(target + ", Histogram")
    plt.subplot(122)
    np.log1p(dataframe[target]).hist(bins=50)
    plt.title(target + ", Logaritmic Histogram")

exam_dependent_variable(df, "Heating_Load")
```



```
exam_dependent_variable(df, "Cooling_Load")
```

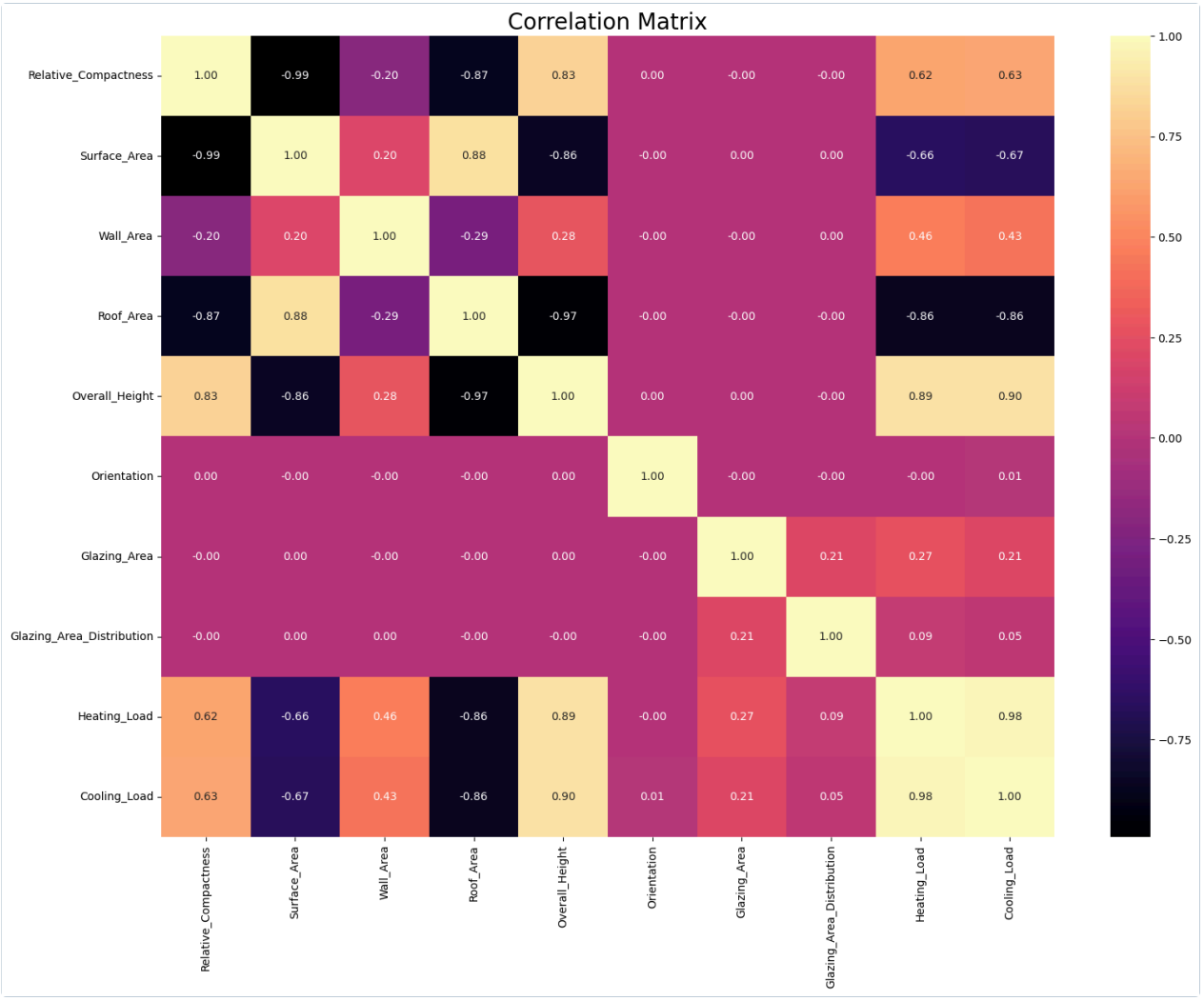


Correlation Analysis

```
df.corr()
```

	Relative_Compactness	Surface_Area	Wall_Area	Roof_Area	Overall_Height	Orientation	Glazing_Area	Glazing_Are
Relative_Compactness	1.000000e+00	-9.919015e-01	-2.037817e-01	-8.688234e-01	8.277473e-01	4.678592e-17	-2.960552e-15	
Surface_Area	-9.919015e-01	1.000000e+00	1.955016e-01	8.807195e-01	-8.581477e-01	-3.459372e-17	3.636925e-15	
Wall_Area	-2.037817e-01	1.955016e-01	1.000000e+00	-2.923165e-01	2.809757e-01	-2.429499e-17	-8.567455e-17	
Roof_Area	-8.688234e-01	8.807195e-01	-2.923165e-01	1.000000e+00	-9.725122e-01	-5.830058e-17	-1.759011e-15	
Overall_Height	8.277473e-01	-8.581477e-01	2.809757e-01	-9.725122e-01	1.000000e+00	4.492205e-17	1.489134e-17	
Orientation	4.678592e-17	-3.459372e-17	-2.429499e-17	-5.830058e-17	4.492205e-17	1.000000e+00	-9.406007e-16	
Glazing_Area	-2.960552e-15	3.636925e-15	-8.567455e-17	-1.759011e-15	1.489134e-17	-9.406007e-16	1.000000e+00	

```
f, ax = plt.subplots(figsize=[18, 13])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=ax, cmap="magma")
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```




```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import Input, Dense
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# Functions for formatting output and normalizing data
def format_output(data):
    y1 = data.pop('Heating_Load')
    y1 = np.array(y1)
    y2 = data.pop('Cooling_Load')
    y2 = np.array(y2)
    return y1, y2

def norm(x):
    return (x - train_stats['mean']) / train_stats['std']
```

```
# Convert output arrays to dictionary format for Keras model
train_Y = {'y1_output': train_Y[0], 'y2_output': train_Y[1]}
test_Y = {'y1_output': test_Y[0], 'y2_output': test_Y[1]}

# Define model layers
input_layer = Input(shape=(len(train.columns),))
first_dense = Dense(units=128, activation='relu')(input_layer)
second_dense = Dense(units=128, activation='relu')(first_dense)

# Y1 output will be fed directly from the second dense
y1_output = Dense(units=1, name='y1_output')(second_dense)
third_dense = Dense(units=64, activation='relu')(second_dense)

# Y2 output will come via the third dense
y2_output = Dense(units=1, name='y2_output')(third_dense)

# Define the model with the input layer and a list of output layers
model = Model(inputs=input_layer, outputs=[y1_output, y2_output])

# Print the model summary
print(model.summary())
```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 8)	0	-
dense_4 (Dense)	(None, 128)	1,152	input_layer_2[0]…
dense_5 (Dense)	(None, 128)	16,512	dense_4[0][0]
dense_6 (Dense)	(None, 64)	8,256	dense_5[0][0]
y1_output (Dense)	(None, 1)	129	dense_5[0][0]
y2_output (Dense)	(None, 1)	65	dense_6[0][0]

Total params: 26,114 (102.01 KB)

Trainable params: 26,114 (102.01 KB)

Non-trainable params: 0 (0.00 B)

None

```
# Compile the model
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss={'y1_output': 'mse', 'y2_output': 'mse'},
              metrics={'y1_output': tf.keras.metrics.RootMeanSquaredError(),
                       'y2_output': tf.keras.metrics.RootMeanSquaredError()})

# Train the model
history = model.fit(norm_train_X, train_Y,
                    epochs=500, batch_size=10, validation_data=(norm_test_X, test_Y))

# Test the model and print loss and mse for both outputs
results = model.evaluate(x=norm_test_X, y=test_Y)

# Print the results to see what is returned
print("Evaluation Results:", results)
```

```
Epoch 401/500
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5135 - y1_output_root_mean_squared_error: 0.4056 - y2_output_root_mean_squared_error: 0.5856WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 0.5128 - y1_output_root_mean_squared_error: 0.4051 - y2_output_root_mean_squared_error: 0.5854 - val_loss: 1.4032 - val_y1_output_root_mean_squared_error: 0.4595 - val_y2_output_root_mean_squared_error: 1.1076
Epoch 402/500
62/62 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.5250 - y1_output_root_mean_squared_error: 0.3959 - y2_output_root_mean_squared_error: 0.6032 - val_loss: 1.6134 - val_y1_output_root_mean_squared_error: 0.5441 - val_y2_output_root_mean_squared_error: 1.1363
Epoch 403/500
54/62 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5506 - y1_output_root_mean_squared_error: 0.3794 - y2_output_root_mean_squared_error: 0.6328WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.5573 - y1_output_root_mean_squared_error: 0.3797 - y2_output_root_mean_squared_error: 0.6385 - val_loss: 1.2253 - val_y1_output_root_mean_squared_error: 0.5113 - val_y2_output_root_mean_squared_error: 0.9873
Epoch 404/500
56/62 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5480 - y1_output_root_mean_squared_error: 0.3449 - y2_output_root_mean_squared_
Uploading artifacts:   0%|          | 0/11 [00:00<?, ?it/s]
Uploading artifacts:   0%|          | 0/2 [00:00<?, ?it/s]
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.5974 - y1_output_root_mean_squared_error: 0.5455 - y2_output_root_mean_squared_error: 1.1365
Evaluation Results: [1.5110019445419312, 0.5072449445724487, 1.113294243812561]
```

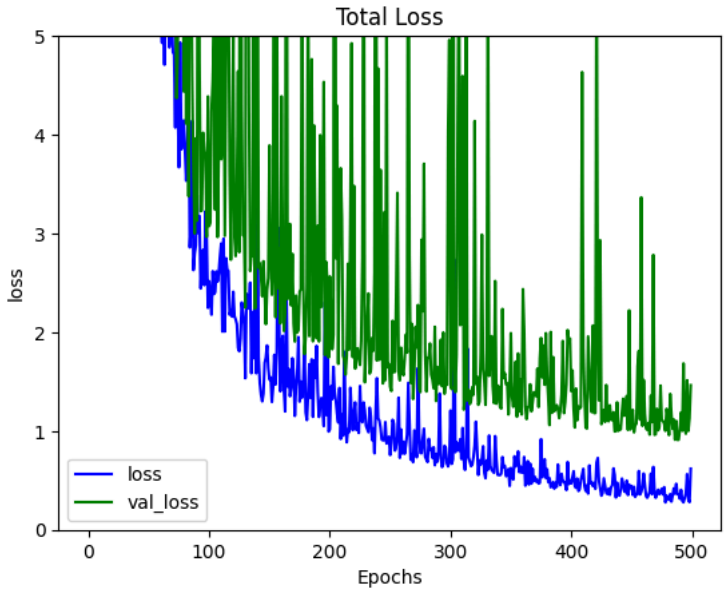
```
# Assuming results has 3 values: total loss, y1_output_rmse, y2_output_rmse
total_loss, y1_rmse, y2_rmse = results
print(f"Total Loss = {total_loss}, Y1 RMSE = {y1_rmse}, Y2 RMSE = {y2_rmse}")
```

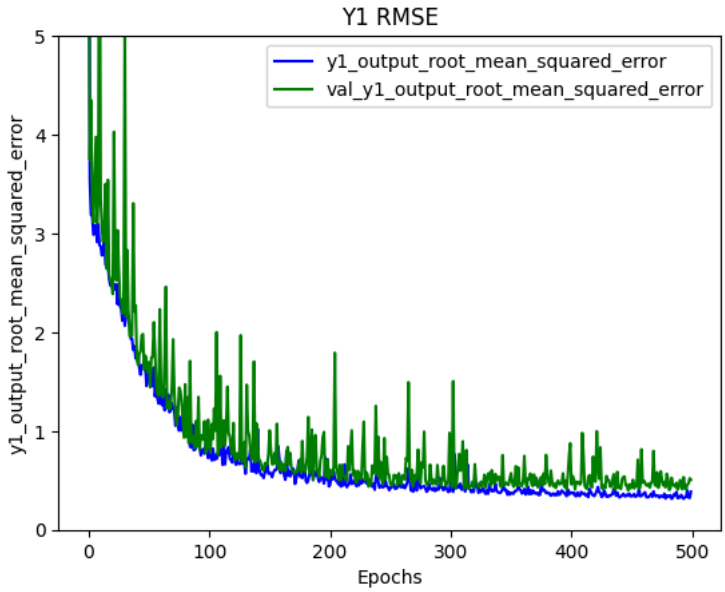
```
Total Loss = 1.5110019445419312, Y1 RMSE = 0.5072449445724487, Y2 RMSE = 1.113294243812561
```

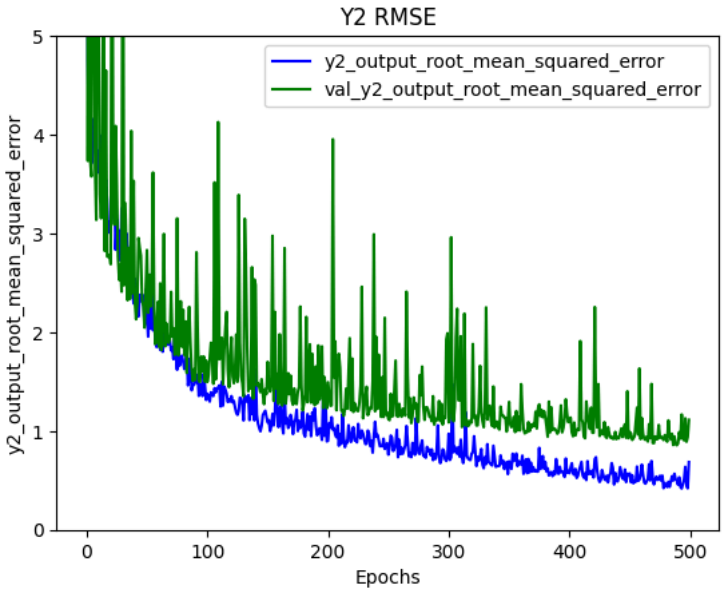
Loss and Metric Over Epochs:

```
# Plotting the training and validation loss and metrics (such as RMSE) over epochs helps you see how well your model is
learning and if it's overfitting.
def plot_metrics(history, metric_name, title, ylim=None):
    plt.figure()
    plt.title(title)
    if ylim:
        plt.ylim(0, ylim)
    plt.plot(history.history[metric_name], color='blue', label=metric_name)
    plt.plot(history.history['val_' + metric_name], color='green', label='val_' + metric_name)
    plt.xlabel('Epochs')
    plt.ylabel(metric_name)
    plt.legend()
    plt.show()

plot_metrics(history, 'loss', 'Total Loss', ylim=5)
plot_metrics(history, 'y1_output_root_mean_squared_error', 'Y1 RMSE', ylim=5)
plot_metrics(history, 'y2_output_root_mean_squared_error', 'Y2 RMSE', ylim=5)
```

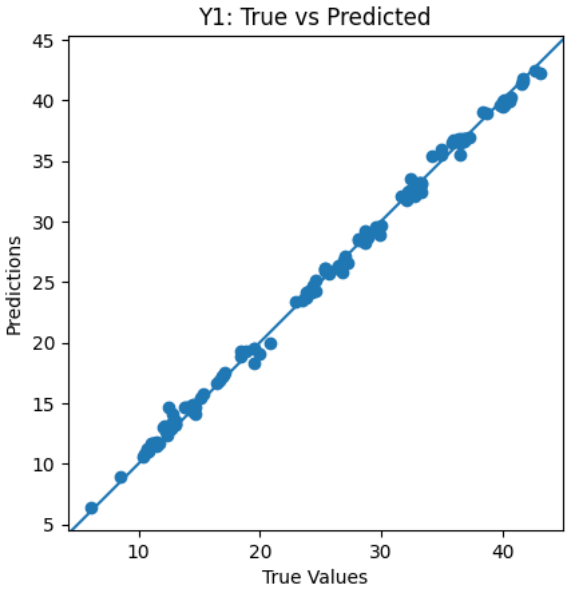


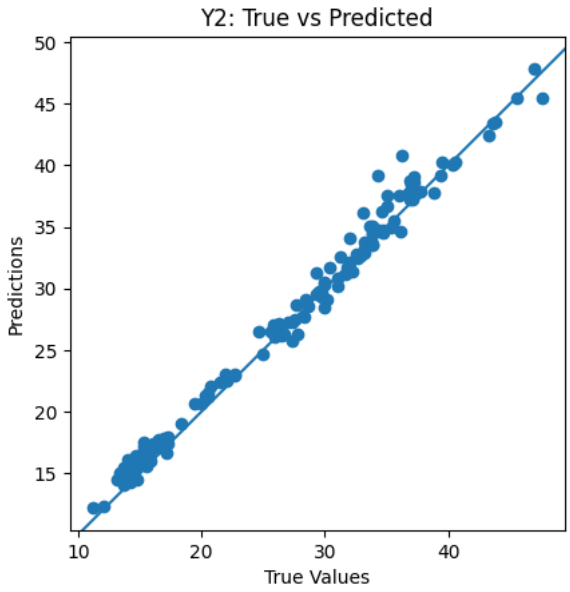




True vs. Predicted Scatter Plot:

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```




Y2: True vs Predicted
Predictions
True Values
50
45
40
35
30
25
20
15
10
20
30
40