## Use Case: ASHARE Data

* + 1. **Read dataset.** Read ASHAE Data.

A screenshot of a computer

Description automatically generated

Figure 2 AHSRAE Dataset

* + 1. **Treat Features.** Split values, take mean value, convert to metric system. Create Dataframe for Specific Heat, Density, Conductivity.

A screenshot of a computer

Description automatically generated

Figure 3 Converted Dataframe

* + 1. **Analyse type of missingness.** Use librabry missingno, msno.matrix(materialData) A black and white bar code

       Description automatically generated

Figure 4 This matrix shows how nulls are scattered across the dataset

A blue and pink squares

Description automatically generated

Figure 5 Heatmap in the form of a correlation matrix, showing where there is a relationship between the missingness inside the data, showing strong correlation between material Density and material Conductivity.

A black and white bar code

Description automatically generated

Figure 6 Sort matrix by either of the column (Material Density or Material Conductivity)

The analysis shows that if a data point is missing in materialDensity, we can guess that it is somewhat related and also missing from materialConductivity and materialHeatCapacity. There is weak correlation between density and capacity. Overall in can be concluded that columns are not missing at random (MNAR).

* + 1. **Imputer Benchmark**

The following benchmarks univariant and multivariant imputations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Method | Significanc | Data Type |
| Mean, Median, Mode |  |  |  |
| KNN Imputer |  |  |  |
| Iterative Imputer |  |  |  |
| K Neighbour Regression |  |  |  |

* + 1. **Mean, Median, Mode**

mean\_imputation = materialData.fillna(materialData.mean())   
median\_imputation = materialData.fillna(materialData.median())   
mode\_imputation = materialData.fillna(materialData.mode().iloc[0])

A graph of colored lines

Description automatically generated

Figure 7 Material Denisty - Mean, Median and Mode imputation

A graph of a graph

Description automatically generated

Figure 8 Material Conductivity -Mean, Median and Mode Imputation

A diagram of a graph

Description automatically generated with medium confidence

Figure 9 Material Heat Capacity - Mean, Median, Mode

* + 1. **KNN Imputer**

import matplotlib.pyplot as plt  
n\_neighbors = [2,3,5,7]  
fig, ax = plt.subplots(figsize=(16, 8))  
# Plot the original distribution  
sns.kdeplot(materialData.materialDensity, label="Original Distribution")  
for k in n\_neighbors:  
 knn\_imp = KNNImputer(n\_neighbors=k)  
 materialData\_knn\_impute.loc[:, :] = knn\_imp.fit\_transform(materialData)  
 sns.kdeplot( materialData\_knn\_impute.materialDensity, label=f"Imputed Dist with k={k}")  
plt.legend();

A graph of colored lines

Description automatically generated

A graph of a line

Description automatically generated

A graph of colored lines

Description automatically generated

* + 1. **Iterative Imputer**

Iterative imputer models each feature with missing values as a function of other features in a round-robin fashion.The ExtraTreesRegressor is used as the estimator. It is a machine-learning model from the ensemble of decision trees. The parameters, #max\_iter = 10 runs on max 10 iterations, #random\_state=1121218 sets the random seed for reproducibility. Initializing the random number generator in a way that ensures that the sequence of random numbers (or stochastic operations) generated will be the same each time the code is run.

ii\_imp = IterativeImputer(  
 estimator=ExtraTreesRegressor(), max\_iter=100, random\_state=1121218)

# Tranform, using [:, :]: This syntax selects all rows and all columns of the DataFrame.  
materialData\_ii\_imputed.loc[:, :] = ii\_imp.fit\_transform(materialData\_ii\_imputed)

# MaterialData\_ii\_imputed.describe()

Coutns the total number of variables per feature (or class). The mean value for material density, conductivity and heat capacity is at 780, 0.36 and 1024 respectively. The standard deviation varies from the mean by 690, 0.69 and 318 for all three classes. The mimum values shows the smallest observed value. The 25%, 50% and 75% indicate the percentage of values performing bellow certain thresshod, as represented inside the Figure 7.

A table with numbers and text

Description automatically generated

Figure 10 Descriptive Statistic of imputed values

plt.scatter(materialData\_ii\_imputed.materialDensity, materialData\_ii\_imputed.materialConductivity)

A graph of blue dots

Description automatically generated

Figure 11 Scatter Plot - Material Density compared to Material Conductivity, where x=density, y=conductivity

plt.scatter(materialData\_ii\_imputed.materialDensity, materialData\_ii\_imputed.materialHeatCapacity)

A graph of blue dots

Description automatically generated

Figure 12 Scatter Plot - Material Density compared to Material Heat Capacity, where x=density, y=heatCapacity

# Plotting---Denisty

fig, ax = plt.subplots(figsize=(16, 8))

# Plot the original distribution

sns.kdeplot(materialData['materialDensity'].dropna(), label="Density Original Distribution", ax=ax)

# Plot iterative imputation density

sns.kdeplot(materialData\_ii\_imputed.materialDensity, label="materialDensity", ax=ax)

# Add legend

plt.legend()

# Add titles and labels

plt.title('Distribution of Material Density Methods')

plt.xlabel('Material Density')

plt.ylabel('Density')

# Show the plot

plt.show()

A graph of a number of blue and orange lines

Description automatically generated

Figure 13 Distribution Material Density Data - Original and imputed (iteratively)

Compare the descriptive statistics, compare mean.

A graph of a normal distribution

Description automatically generated

Figure 14 Standard Deviation and Mean comparison - Original and imputed (iteratively)

Plot material conductivity and heat capacity data distribution, original compared to imputed.

A graph of a person

Description automatically generated

Figure 15 Distribution Material Conductivity Data - Original and imputed (iteratively)

A graph of a graph

Description automatically generated with medium confidence

Figure 16 Distribution Material Heat Capacity Data - Original and imputed (iteratively)

* + 1. **Benchmark KNN and Iterative Impute**

In the following the IterativeImputer by Sklearn: <https://scikit-learn.org/stable/auto_examples/impute/plot_iterative_imputer_variants_comparison.html> will be studied, comparing the BayesianRidge and ExtraTreeRegressor, as they yield the best results.

Predict materialDensity based on materialConductivity and materialHeatCapacity, suit regression involve imputing missing values in dataset, training regression models, and evaluating their performance using cross-validation. In the first step the data preparations separates the features (materialConductivity and materialHeatCapacity) and the target (materialDensity). Second step prepares the imputations and regressions. Use imputation methods like **IterativeImputer** and **KNNImputer**. Use regression models like **BayesianRidge and ExtraTreesRegressor**. Third step evaluates the performance of the models with a cross-validation.

ii\_scores = pd.DataFrame()

for estimator in [BayesianRidge(), ExtraTreesRegressor()]:

pipe = make\_pipeline(

IterativeImputer(estimator=estimator, random\_state=state), estimator)

ii\_scores[estimator.\_\_class\_\_.\_\_name\_\_] = cross\_val\_score(

pipe, X\_full, y\_full.ravel(), scoring=scoring, cv=cv)

**BayesianRidge** is a linear model, making it suitable for datasets where relationships between variables are linear or approximately linear.

**ExtraTreesRegressor** is a non-linear ensemble model that can

handle more complex and non-linear relationships in the data. Bayesian Ridge might provide smoother imputations based on linear relationships, whereas Extra Trees might handle more complex data distributions.

# Store KNN scores

knn\_scores = pd.DataFrame()

n\_neighbors = [2, 3, 5, 7, 9]

for k in n\_neighbors:

pipe = make\_pipeline(KNNImputer(n\_neighbors=k), BayesianRidge())

knn\_scores[f"KNN(k = {k})"] = cross\_val\_score(

pipe, X\_full, y\_full.ravel(), scoring=scoring, cv=cv )

A graph with blue and black bars

Description automatically generated

Figure 17 Cross Validation for Benchmarking KNN (2, 3, 5, 7, 9) BayesianRidge, ExtraTreesRegressor

## Regression

Linearly modelling the relationship between a scalar response and one or more explanatory variable (also known as dependent and independent variables).

The materialData\_ii\_imputed is used as input data derived from the iterative imputation. It uses ExtraTreesRegressor as methods, 100 iterations and a random seed at 1121218.

#Define dependent and independent variables  
Y = np.array(materialData\_ii\_imputed.materialDensity).reshape(-1,1)  
X = np.array(materialData\_ii\_imputed[['materialConductivity','materialHeatCapacity']])

#### Split data use Shuffle Split

The method shuffle split allows to assess the model's performance across different splits of the data, which can provide a more robust evaluation of the model.

#Create a ShuffleSplit object:

rs = ShuffleSplit(n\_splits=5, test\_size=.25, random\_state=0)

n\_splits=5: Number of re-shuffling & splitting iterations.

test\_size=0.25: Proportion of the dataset to include in the test split (25%).

random\_state=0: Seed used by the random number generator to ensure reproducibility.

Print the ShuffleSplit object:

rs.get\_n\_splits(X)

rs.get\_n\_splits(Y)

Generate and print train/test indices for each split: multiple feature matrices (e.g., X1,X2,X3), combine to a single feature matrix before splitting the data into training and testing sets.

for i, (train\_index, test\_index) in enumerate(rs.split(X, Y)):

print(f"Fold {i}:")

print(f" Train: index={train\_index}")

print(f" Test: index={test\_index}")

Y\_train, Y\_test = Y[train\_index], Y[test\_index]

X\_train, X\_test = X[train\_index], X[test\_index]

print(f" X\_train: \n{X\_train}")

print(f" X\_test: \n{X\_test}")

print(f" Y\_train: {Y\_train}")

print(f" Y\_test: {Y\_test}")

#### Linear Regression

reg = LinearRegression().fit(X\_train, Y\_train)

print(reg.score(X\_train, Y\_train))

print('coef slope', reg.coef\_)

print('intercept\_final data point for x and y', reg.intercept\_)

print(reg.score)

**Results**

standard coefficient 0.6152352048975964

coef slope [[593.07336 -0.564490632]]

intercept\_final data point for x and y [ 1308.24176157]]

**Interpretation**

**The R-squared score** (coefficient of determination) measures the proportion of the variance in the dependent variable (Y\_train) that is predictable from the independent variables (X\_train). n R-squared value of 0.615 means that approximately 61.5% of the variance in materialDensity (dependent variable) is explained by the features materialHeatCapacity and materialConductivity (independent variables). The remaining 38.5% of the variance is due to other factors not included in the model.

**The coefficients** represent the change in the dependent variable (materialDensity) for a **one-unit change** in each independent variable, holding all other variables constant

**materialConductivity** (593.07336 ), for every one unit increase in materialConductivity, materialDensity decreases by approximately 0.564 units, assuming materialHeatCapacity remains constant.

**materialHeatCapacity** (-0.564490632) for every one unit increase in materialHeatCapacity, materialDensity increases by approximately 593.07 units, assuming materialConductivity remains constant.

The **Interpretation** is the expected value of the dependent variable (materialDensity) when all independent variables are zero. **When** both materialHeatCapacity and materialConductivity are zero, the expected materialDensity is approximately 1308.24 units

**Regression formula:**

**Regression plot**

**Predicted Density (Y) based on Material Conductivity (X1)**

#predicted density in relation to conductivity, applying log-linear regression

sns.regplot(data=materialData\_ii\_imputed, y=reg.predict(X\_test), x= X\_test[:,0], logx=True)

>> Why Log linear regression: A log-linear model is a [mathematical model](https://en.wikipedia.org/wiki/Mathematical_model) that takes the form of a [function](https://en.wikipedia.org/wiki/Function_(mathematics)) whose [logarithm](https://en.wikipedia.org/wiki/Logarithm) equals a [linear combination](https://en.wikipedia.org/wiki/Linear_combination) of the [parameters](https://en.wikipedia.org/wiki/Parameter#Mathematical_models) of the model, which makes it possible to apply (possibly [multivariate](https://en.wikipedia.org/wiki/Multivariate_analysis)) [linear regression](https://en.wikipedia.org/wiki/Linear_regression). That is, it has the general form of in which the fi(X) are quantities that are functions of the variable X, in general a vector of values, while c and the wi stand for the model parameters.

# Adding titles and labels

plt.title('Regression Prediction - Density Predict on Material Conductivity')

plt.xlabel('Material Conductivity X1')

plt.ylabel('Predicted Density Y')

plt.legend()

A graph of a graph showing the difference between material conductivity and material conductivity

Description automatically generated

**Predicted Density (Y) based on Material Heat Capacity (X2)**

sns.regplot(data=materialData\_ii\_imputed, y=reg.predict(X\_test), x= X\_test[:,1], logx=True) #predicted density in relation to heat capacity

# Adding titles and labels

plt.title('Regression Prediction - Density Predict on Material Heat Capacity')

plt.xlabel('Material Heat Capacity X1')

plt.ylabel('Predicted Density Y')

plt.legend()

A graph of heat capacity

Description automatically generated with medium confidence

#### Validation

Result

test\_score = reg.score(X\_test, Y\_test)

print("Test R^2 score:", test\_score)

Test R^2 score: 0.6557736109366549

Since the dataset was split in training and testing data, the predicted\_Y\_for\_X\_test predicts the Y based on the test data, using the coefficience of the variables in the X\_train data (x1= conducity and x2=heat capacity). The following compares the predicted\_Y\_for\_X\_test with the actual Y values from the Y\_train dataset.

**Loss function formula:**

Two commonly used loss functions are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE).

MSE measures the average of the squared differences between predicted and actual values, see Formular below. The **advantage** is that it offers faster convergence in scenarios where the error values are relatively small and consistent. A ML model reaches convergence when it achieves a state during training in which [loss](https://machine-learning.paperspace.com/wiki/accuracy-and-loss#loss) settles to within an error range around the final value. In other words, a model converges when additional training will not improve the model. The key to its rapid convergence lies in the error amplification mechanism it employs. The squared term magnifies the impact, which accelerates the minimization process during training. The **disadvantage** is that MSE treats all errors with equal importance, which means outliers have a substantial impact on the loss calculation

Mean Squared Error: 162589.91521600925

Mean Absolute Error: 312.31993496279347

|  |
| --- |
| Mean Squared Error (MSE) is a statistical measure used to evaluate the quality of a model by quantifying the difference between actual and predicted values. |