GNANAMANI COLLEGE OF TECHNOLOGY

COLLEGE CODE: 6208

DEPAPTMENT OF ELECTRONIC AND COMMUNICATION

ENGINEERING

III- YEAR

TOPIC NAME : MEASURE ENERGY CONSUMPTION

PRESENTED BY

Sivanesan.P

**MEASURE ENERGY CONSUMPTION**

**PHASE-1**

**PROBLEM:**

Design thinking is a problem-solving approach that emphasizes empathy, creativity, and iterative prototyping. When applying design thinking to measure energy consumption, consider the following steps:

# Empathize:

Understand the needs and perspectives of users. Talk to individuals or organizations who are interested in monitoring their energy usage. What are their pain points, goals, and challenges?

# Define:

Clearly articulate the problem. In this case, it might be something like, "How can we create a user-friendly system to accurately measure and visualize energy consumption?"

# Ideate:

Generate a wide range of ideas. Think creatively about different methods and technologies that could be used to measure energy consumption. Consider smart meters, IoT sensors, data visualization tools, etc.

# Prototype:

Create a simplified version of your solution. It could be a mockup, a basic sensor setup, or a digital interface for viewing energy data.

# Test:

Get feedback on your prototype. Test it with potential users to see how well it meets their needs. Does it provide accurate data? Is it user-friendly? Are there any pain points

Based on feedback, refine your prototype. Make improvements and test again. Repeat this process as many times as necessary to create an effective solution.

# Implement:

Develop the final version of your energy consumption measurement system. Ensure its user-friendly, accurate, and meets the needs identified during the empathy phase.

# Evaluate:

After implementation, monitor the performance of your system. Is it delivering the expected results? Are users satisfied with it? Gather feedback for potential future enhancements.

Remember, design thinking is a flexible framework, so adapt these steps to suit the specific context and needs of your project. Additionally, consider sustainability and efficiency in the design of your energy measurement system to align with the goal of reducing energy consumption.

**PHASE-2**

**Innovation:**

* Investment vs. Savings: Analyze the return on investment (ROI) of energy-saving innovations by comparing the initial cost with the long-term energy cost savings.
* Carbon Emissions Reduction: Measure the reduction in carbon emissions achieved through energy-saving innovations, using emission factors and data on energy consumption.
* Adoption Rate: Track the adoption rate of innovative energy-saving technologies and practices in a specific industry or region to gauge their impact on overall energy consumption.
* Energy Efficiency Ratings: Evaluate products and technologies using standardized energy efficiency ratings, such as ENERGY STAR for appliances or LEED certification for buildings.
* Energy Savings Calculations: Calculate the energy savings achieved by innovative technologies compared to traditional counterparts. This can involve before-and-after assessments or modeling based on usage data.
* Government Policies and Incentives: Evaluate the impact of government policies, incentives, and subsidies on encouraging energy-efficient innovation and adoption.
* Industry Standards: Monitor the development and adoption of industry standards related to energy efficiency, which can drive innovation.
* Consumer Awareness: Measure consumer awareness and willingness to adopt energy-efficient products and practices through surveys and market research.
* Case Studies and Pilot Programs: Analyze real-world case studies and pilot programs to understand the practical implications and success stories of energy-saving innovations.

**CODE USING PHYTHON :**

import sensor\_library # Replace with the actual library for your sensor

# Initialize the sensor

sensor = sensor\_library.initialize\_sensor()

# Initialize variables start\_time = time.time()

total\_energy = 0 # Initialize total energy to 0

# Main loop to read data and calculate energy consumption while True:

# Read data from the sensor data = sensor.read\_data()

# Calculate elapsed time current\_time = time.time()

elapsed\_time = current\_time - start\_time

# Calculate power consumption (replace with appropriate formula for your sensor) power = data['voltage'] \* data['current']

# Calculate energy consumption energy = power \* elapsed\_time

# Update total energy total\_energy += energy

# Reset start time start\_time = current\_time

# Print or store the total energy consumption

print(f"Total Energy Consumption: {total\_energy} Joules")

# Sleep for a while before the next reading time.sleep(60) # Sleep for 60 seconds (adjust as needed)

# Main loop to read data and calculate energy consumption while True:

# Read data from the sensor data = sensor.read\_data()

# Calculate elapsed time current\_time = time.time()

elapsed\_time = current\_time - start\_time

# Calculate power consumption (replace with appropriate formula for your sensor) power = data['voltage'] \* data['current']

# Calculate energy consumption energy = power \* elapsed\_time

# Update total energy total\_energy += energy

# Reset start time start\_time = current\_time

# Print or store the total energy consumption

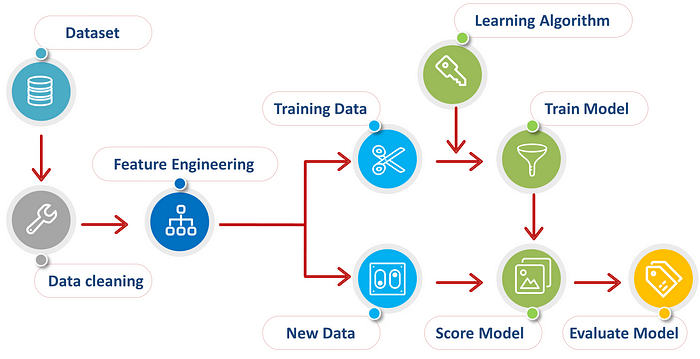
print(f"Total Energy Consumption: {total\_energy} Joules")

# Sleep for a while before the next reading time.sleep(60) # Sleep for 60 seconds (adjust as needed)

**PHASE 3**

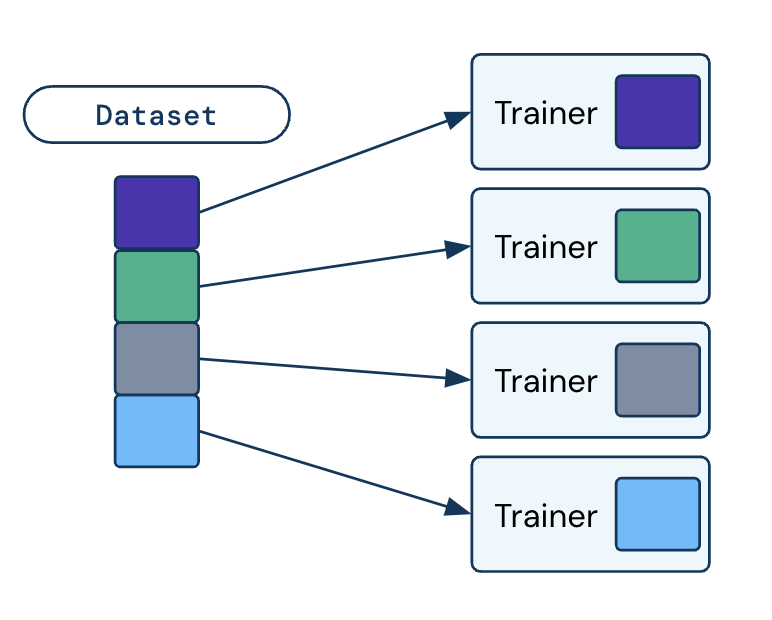
**DEVELOPMENT PART:**

In order to derive knowledge and insights from data, the area of data science integrates statistical analysis, machine learning, and computer programming. It entails gathering, purifying, and converting unstructured data into a form that can be analysed and visualised.

**Dataset**

If the dataset already contains rows, the incoming data from the data source is merged with the existing rows. The Load method can be used in several common scenarios, all centered around getting data from a specified data source and adding it to the current data container (in this case, a dataset ).

Get\_dataset(dataset\_slug, dataset\_type [, partition, split, split\_type, transform])



Input

----------

Dataset\_slug: str

Slug of the dataset to retrieve

Dataset\_type: str

The type of dataset [classification, instance-segmentation, semantic-segmentation]

Partition: str

Selects one of the partitions [train, val, test, None]. (Default: None)

Split: str

Selects the split that defines the percentages used. (Default: 'default')

Split\_type: str

Heuristic used to do the split [random, stratified]. (Default: 'random')

Transform : list[torchvision.transforms]

List of pytorch transforms. (Default: None)

**Output:**

----------

Dataset: localdataset

API class to the local dataset

From darwin.torch import get\_dataset

Dataset\_id = "v7-demo/bird-species"

Dataset = get\_dataset(dataset\_id, dataset\_type="instance-segmentation")

Print(dataset)

# Returns:

# instancesegmentationdataset():

# Root: /home/jon/.darwin/datasets/v7-demo/bird-species

# Number of images: 1909

# Number of classes: 3

From darwin.torch import get\_dataset

Import darwin.torch.transforms as T

Dataset\_id = "v7-demo/bird-species"

Trfs\_train = T.Compose([T.randomhorizontalflip(), T.totensor()])

Db\_train = get\_dataset(dataset\_id, dataset\_type="instance-segmentation", \

Partition="train", split\_type="stratified", transform=trfs\_train)

Trfs\_val = T.totensor()

Db\_val = get\_dataset(dataset\_id, dataset\_type="instance-segmentation", \

Partition="val", split\_type="stratified", transform=trfs\_val)

Print(db\_train)

# Returns:

# instancesegmentationdataset():

# Root: /home/jon/.darwin/datasets/v7-demo/bird-species

# Number of images: 1336

# Number of classes: 3

**Different Ways to Load Data in Python:**

**1. Manual Function**

This is the most difficult, as you have to design a custom function, which can load data for you. You have to deal with Python’s normal filing concepts and using that you have to read a .csv file.

Let’s do that on 100 Sales Records file.

Def load\_csv(filepath):

Data = []

Col = []

Checkcol = False

With open(filepath) as f:

For val in f.readlines():

Val = val.replace("\n","")

Val = val.split(',')

If checkcol is False:

Col = val

Checkcol = True

Else:

Data.append(val)

Df = pd.dataframe(data=data, columns=col)

Return df

**Output**

Mydata = load\_csv('100 Sales Record.csv')

Print(mydata.head())

**2. Numpy.loadtxt function:**

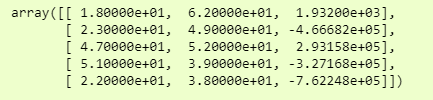
This is a built-in function in Numpy, a famous numerical library in Pyth

To get the data of a single type, you can download [this](https://docs.google.com/spreadsheets/d/16mgiYbNz-XaW_r6GXUy2cJ0hy2E-lxwFLaVXAYIAOj0/edit?usp=sharing) dummy dataset. Let’s jump to code.

Df = np.loadtxt('convertcsv.csv', delimeter = ',')

Now if we print df, we will see our data in pretty decent numpy arrays that are ready to use.

Print(df[:5,:])

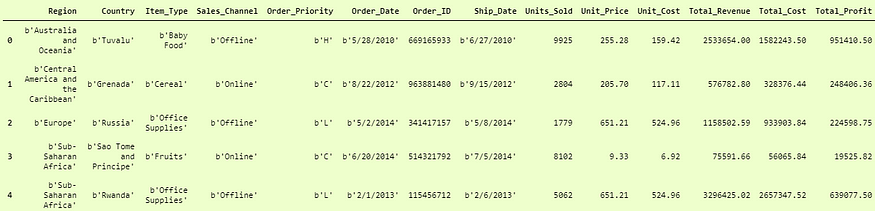


**3. Numpy.genfromtxt():**

Data = np.genfromtxt('100 Sales Records.csv', delimiter=',', dtype=None, names = True)

And we can print it as:

>>> pd.dataframe(df3).head()



**4. Pandas.read\_csv():**

Pandas is a very popular data manipulation library, and it is very commonly used. One of it’s very important and mature functions is read\_csv() which can read any .csv file very easily and help us manipulate it. Let’s do it on our 100-Sales-Record dataset.

This function is very popular due to its ease of use. You can compare it with our previous codes, and you can check it.

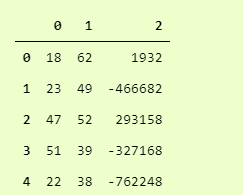
>>> pddf = pd.read\_csv('100 Sales Record.csv')

>>> pddf.head()

Example in our convertcsv.csv file, we had no column names so we can read it as:

>>> newdf = pd.read\_csv('convertcsv.csv', header=None)

>>> newdf.head()



**5. Pickle:**

When your data is not in a good, human-readable format, you can use pickle to save it in a binary format. Then you can easily reload it using the pickle library.

We will take our 100-Sales-Record CSV file and first save it in a pickle format so we can read it.

With open('test.pkl','wb') as f:

Pickle.dump(pddf, f)

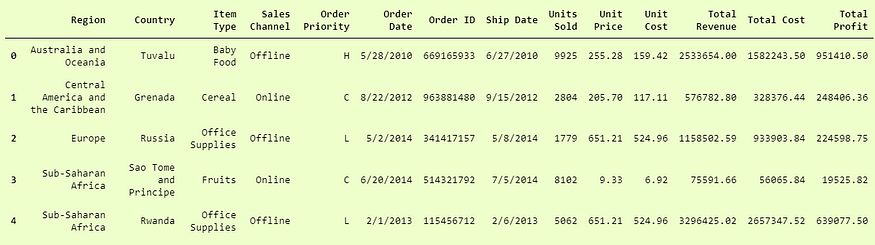
This will create a new file test.pkl which has inside it our pddf from Pandas heading.

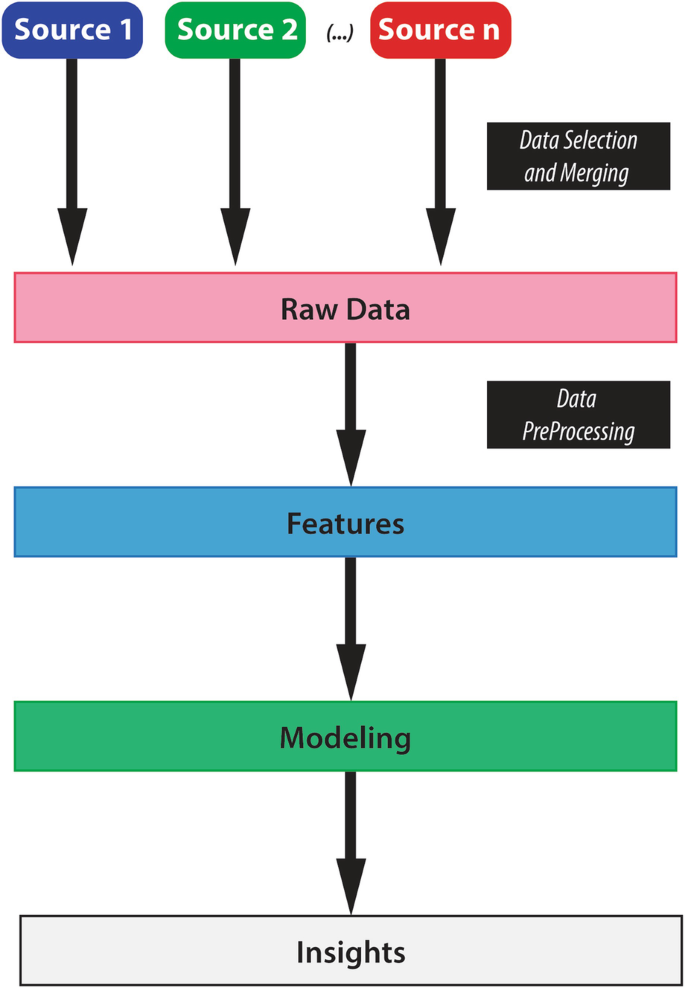
Now to open it using pickle, we just have to use pickle.load function.

With open("test.pkl", "rb") as f:

D4 = pickle.load(f)

>>> d4.head()





Data Loading and Preprocessing

Ray Train integrates with [Ray Data](https://docs.ray.io/en/latest/data/data.html#data) to offer an efficient, streaming solution for loading and preprocessing large datasets. We recommend using Ray Data for its ability to performantly support large-scale distributed training workloads.

In this guide, we will cover how to incorporate Ray Data into your Ray Train script, and different ways to customize your data ingestion pipeline.

**Install Ray Data and Ray Train:**

Pip install -U "ray[data,train]"

Data ingestion can be set up with four basic steps:

Create a Ray Dataset.

Preprocess your Ray Dataset.

Input the preprocessed Dataset into the Ray Train Trainer.

Consume the Ray Dataset in your training function.

**How to Preprocess Data in Python Step-by-Step:**

* Load data in Pandas.
* Drop columns that aren't useful.
* Drop rows with missing values.
* Create dummy variables.
* Take care of missing data.
* Convert the data frame to numpy.
* Divide the data set into training data and test data.

**1. Load Data in Pandas:**

To work on the data, you can either load the CSV in Excel or in [Pandas](https://builtin.com/data-science/data-wrangling-pandas). For the purposes of this tutorial, we’ll load the CSV data in Pandas.

Df = pd.read\_csv('train.csv')

Let's take a look at the data format below:

>>> df.info()

<class 'pandas.core.frame.dataframe'>

Int64Index: 891 entries, 0 to 890

Data columns (total 12 columns):

Passengerid 891 non-null int64

Survived 891 non-null int64

Pclass 891 non-null int64

Name 891 non-null object

Sex 891 non-null object

Age 714 non-null float64

Sibsp 891 non-null int64

Parch 891 non-null int64

Ticket 891 non-null object

Fare 891 non-null float64

Cabin 204 non-null object

Embarked 889 non-null object

**2. Drop Columns That Aren’t Useful:**

Let's try to drop some of the columns which won't contribute much to our machine learning model. We’ll start with Name, Ticket and Cabin.

Cols = ['Name', 'Ticket', 'Cabin']

Df = df.drop(cols, axis=1)

We dropped three columns:

>>>df.info()

Passengerid 891 non-null int64

Survived 891 non-null int64

Pclass 891 non-null int64

Sex 891 non-null object

Age 714 non-null float64

Sibsp 891 non-null int64

Parch 891 non-null int64

Fare 891 non-null float64

Embarked 889 non-null object

**3. Drop Rows With Missing Values:**

we can drop all rows in the data that have missing values (nans).

>>> df = df.dropna()

>>> df.info()

Int64Index: 712 entries, 0 to 890

Data columns (total 9 columns):

Passengerid 712 non-null int64

Survived 712 non-null int64

Pclass 712 non-null int64

Sex 712 non-null object

Age 712 non-null float64

Sibsp 712 non-null int64

Parch 712 non-null int64

Fare 712 non-null float64

Embarked 712 non-null object

**4. Creating Dummy Variables:**

Instead of wasting our data, let’s convert the Pclass, Sex and Embarked to columns in Pandas and drop them after conversion.

Dummies = []

Cols = ['Pclass', 'Sex', 'Embarked']

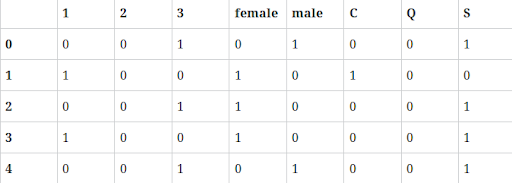
For col in cols:

Dummies.append(pd.get\_dummies(df[col]))

Then…

Titanic\_dummies = pd.concat(dummies, axis=1)

Now we’ve transformed  eight columns wherein 1, 2 and 3 represent the passenger class.



Finally we concatenate to the original data frame, column-wise:

Df = pd.concat((df,titanic\_dummies), axis=1)

Now that we converted Pclass, Sexand Embarked values into columns, we drop the redundant columns from the data frame.

Df = df.drop(['Pclass', 'Sex', 'Embarked'], axis=1)

Let's take a look at the new data frame:

Let's take a look at the new data frame:

>>>df.info()

Passengerid 891 non-null int64

Survived 891 non-null int64

Age 714 non-null float64

Sibsp 891 non-null int64

Parch 891 non-null int64

Fare 891 non-null float64

1 891 non-null float64

2 891 non-null float64

3 891 non-null float64

Female 891 non-null float64

Male 891 non-null float64

C 891 non-null float64

Q 891 non-null float64

S 891 non-null float64

**5. Take Care of Missing Data:**

Let’s compute a median or interpolate() all the ages and fill those missing age values. Pandas has an interpolate() function that will replace all the missing nans to interpolated values.

Df['Age'] = df['Age'].interpolate()

Now let's observe the data columns. Notice Ageis now interpolated with imputed new values.

>>>df.info()

Data columns (total 14 columns):

Passengerid 891 non-null int64

Survived 891 non-null int64

Age 891 non-null float64

Sibsp 891 non-null int64

Parch 891 non-null int64

Fare 891 non-null float64

1 891 non-null float64

2 891 non-null float64

3 891 non-null float64

Female 891 non-null float64

Male 891 non-null float64

C 891 non-null float64

Q 891 non-null float64

**6. Convert the Data Frame to numpy:**

Now that we’ve converted all the data to integers, it's time to prepare the data for machine learning models. Thi

X= Input set with 14 attributes

Y = Small y output, in this case Survived

Now we convert our data frame from Pandas to [numpy](https://builtin.com/data-science/numpy-random-seed) and we assign input and output:

X = df.values

Y = df['Survived'].values

X still has Survived values in it, which should not be there. So we drop in the numpy column, which is the first column.

X = np.delete(X, 1, axis=1)

**7. Divide the Data Set Into Training Data and Test Data:**

Now that we’re ready with X and y, let's split the data set: we’ll allocate 70 percent for training and 30 percent for tests using [scikit](https://builtin.com/machine-learning/scikit-learn-guide) model\_selection.

From sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

**Conclusion:**

preprocessing are essential steps in data preparation, involving data loading, cleaning, transformation, and splitting. These steps are crucial to ensure data quality and suitability for analysis or machine learning.

**PHASE 4**

**Feature Engineering**

Feature engineering is a machine learning technique that leverages data to create new variables that aren’t in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of **simplifying and speeding up data transformations** while also **enhancing model accuracy**. Feature engineering is required when working with machine learning models. Regardless of the data or architecture, a terrible feature will have a direct impact on your model.

**1. Utility Bills:** For residential and commercial energy consumption, you can refer to utility bills, such as electricity, gas, or water bills, which provide information on usage and costs.

**2. Smart Meters:** Many modern homes and businesses are equipped with smart meters that provide real-time data on energy consumption. These meters often offer more detailed insights and can be monitored remotely.

**3. Submeters:** In larger buildings or industrial settings, submeters can be installed to measure energy usage for specific areas, equipment, or processes separately.

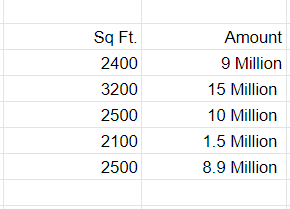
**4. Energy Monitoring Systems:** These systems use sensors and software to continuously monitor and record energy usage. They are often used in industrial and commercial settings for detailed analysis.

**5. Data Loggers:** Data loggers are portable devices that can be attached to specific equipment or systems to record energy consumption over time. They are often used for research or auditing purposes.

**6. Iot Devices:** Internet of Things (iot) devices can be deployed to monitor energy consumption in real-time and provide data for analysis and control, often used in home automation.

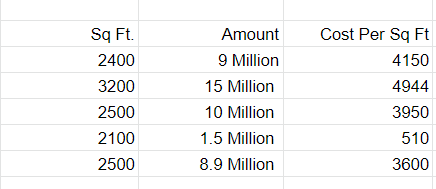
**7. Energy Audits:** Professional energy auditors use various tools and inspections to assess and measure energy consumption in buildings and facilities.

Now to understand it in a much easier way, let’s take a **simple example**. Below are the prices of properties in x city. It shows the area of the house and total price.

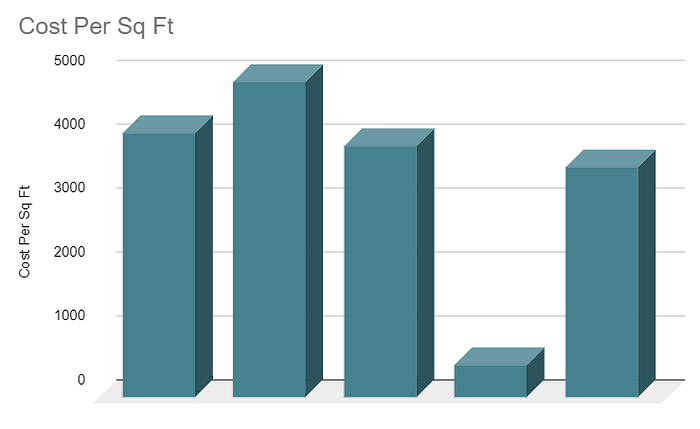


Sample Data

Now this data might have some errors or might be incorrect, not all sources on the internet are correct. To begin, we’ll add a new column to display the cost per square foot.



This new feature will help us understand a lot about our data. So, we have a new column which shows cost per square ft. There are **three main ways** you can find any error. You can use **Domain Knowledge** to contact a property advisor or real estate agent and show him the per square foot rate. If your counsel states that pricing per square foot cannot be less than 3400, you may have a problem. The data can be **visualised**.



When you plot the data, you’ll notice that one price is significantly different from the rest. In the **visualisation method**, you can readily notice the problem. The third way is to use **Statistics**to analyze your data and find any problem. Feature engineering consists of various process -

* **Feature Creation**: Creating features involves creating new variables which will be most helpful for our model. This can be adding or removing some features. As we saw above, the cost per sq. Ft column was a feature creation.
* **Transformations**: Feature transformation is simply a function that transforms features from one representation to another. The goal here is to plot and visualise data, if something is not adding up with the new features we can reduce the number of features used, speed up training, or increase the accuracy of a certain model.
* **Feature Extraction**: Feature extraction is the process of extracting features from a data set to identify useful information. Without distorting the original relationships or significant information, this compresses the amount of data into manageable quantities for algorithms to process.
* **Exploratory Data Analysis:**Exploratory data analysis (EDA) is a powerful and simple tool that can be used to improve your understanding of your data, by exploring its properties. The technique is often applied when the goal is to create new hypotheses or find patterns in the data. It’s often used on large amounts of qualitative or quantitative data that haven’t been analyzed before.
* **Benchmark**: A Benchmark Model is the most user-friendly, dependable, transparent, and interpretable model against which you can measure your own. It’s a good idea to run test datasets to see if your new machine learning model outperforms a recognised benchmark. These benchmarks are often used as measures for comparing the performance between different machine learning models like neural networks and support vector machines, linear and non-linear classifiers, or different approaches like bagging and boosting. To learn more about feature engineering steps and
* process, check the links provided at the end of this article. Now, let’s have a look at why we need feature engineering in machine learning.

**Import pandas as pd:**

From sklearn.model\_selection import train\_test\_split

From sklearn.preprocessing import standardscaler

From sklearn.feature\_selection import selectkbest, f\_classif

From sklearn.decomposition import PCA

# Load your dataset, replace 'your\_data.csv' with the actual file path

Data = pd.read\_csv('your\_data.csv')

# Split the data into features and target variable

X = data.drop('target', axis=1)

Y = data['target']

# Data Preprocessing

# Handle missing values, scaling, and other preprocessing steps here

# Example: X = X.fillna(0) # Replace missing values with zeros

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature Selection

# Example: Select the top 5 features based on ANOVA F-value

Selector = selectkbest(score\_func=f\_classif, k=5)

X\_train = selector.fit\_transform(X\_train, y\_train)

X\_test = selector.transform(X\_test)

# Feature Transformation

# Example: Apply Principal Component Analysis (PCA)

Pca = PCA(n\_components=2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

# Now, X\_train and X\_test contain the engineered features that you can use for modeling

# You can proceed to build and train your machine learning model using the engineered features

# Don't forget to assess the performance of your model as well

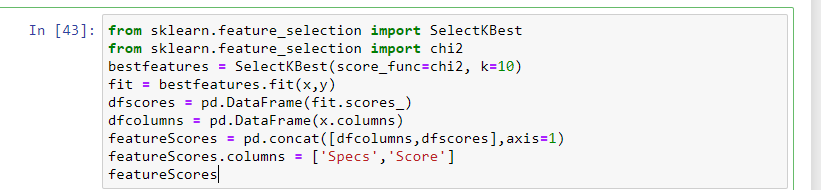
# Ensure you import the necessary libraries before running this code

 Univariate Selection

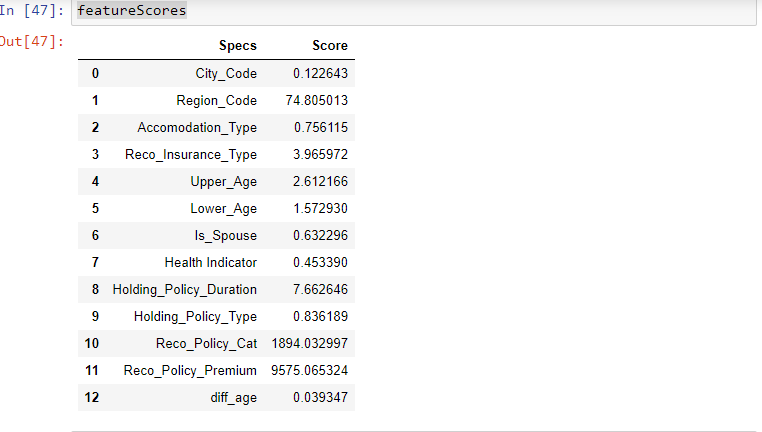
In this, Statistical tests can be used to select the independent features which have the strongest relationship with the dependent feature. [Selectkbest](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) method can be used with a suite of different statistical tests to select a specific number of

Features

INPUT :



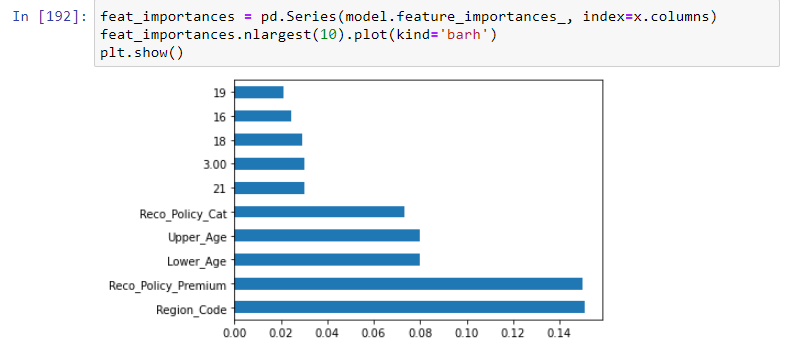
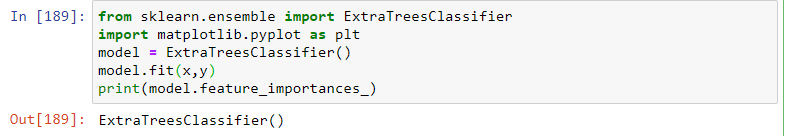
OUTPUT:



 Which feature has the highest score will be more related to the dependent feature and choose those features for the model.

Extratreesclassifier method:

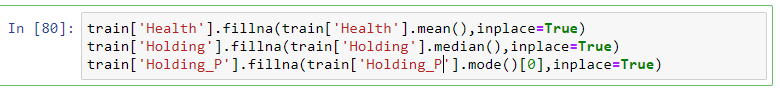
In this method, the extratreesclassifier method will help to give the importance of each independent feature with a dependent feature. Feature importance will give you a score for each feature of your data, the higher the score more important or relevant to the feature towards your output variable.



3. Handling Missing Values

In some datasets, we got the NA values in features. It is nothing but missing data. By handling this type of data there are many ways:

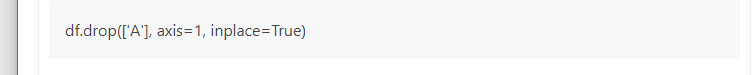
In the missing value places, to replace the missing values with mean or median to numerical data and for categorical data with mode.



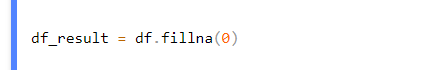
Drop NA values entire rows.

Description: Drop NA

 Drop NA values entire features. (it helps if NA values are more than 50% in a feature)



 Replace NA values with 0.



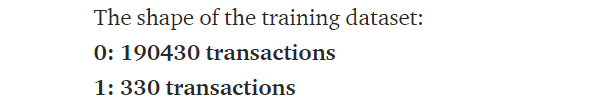
If you choose to drop options, there is a chance to lose important information from them. So better to choose to replace options.

 4. Handling imbalanced data

Why need to handle imbalanced data? Because of to reduce overfitting and underfitting problem.

*Suppose*a feature has a factor level2(0 and 1). It consists of 1’s is 5% and 0’s is 95%. It is called imbalanced data.

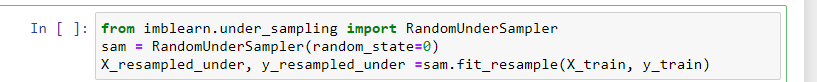
Example:-



By preventing this problem there are some methods:

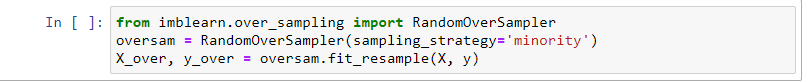
Under-sampling majority class:

Under-sampling the majority class will resample the majority class points in the data to make them equal to the minority class.



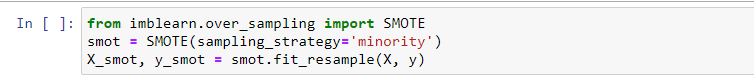
Over Sampling Minority class by duplication:

Oversampling minority class will resample the minority class points in the data to make them equal to the majority class.



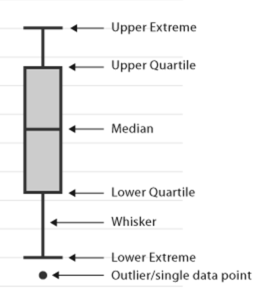
Over Sampling minority class using Synthetic Minority Oversampling Technique (SMOTE):

In this method, synthetic samples are generated for the minority class and equal to the majority class.



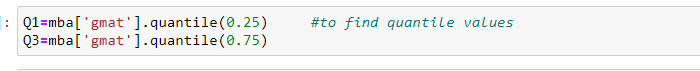
5. Handling outliers

Firstly, calculate the skewness of the features and check whether they are positively skewed, negatively skewed, or normally skewed. Another method is to plot the boxplot to features and check if any values are out of bounds or not. If there, they are called outliers.



How to handle these outliers: –

First, calculate quantile values at 25% and 75%.



  Calculate the Interquartile range

IQR = Q3 – Q1



Calculate the upper extreme and lower extreme values

Lower extreme=Q1 – 1.5 \* IQR

Upper extreme=Q3– 1.5 \* iqre

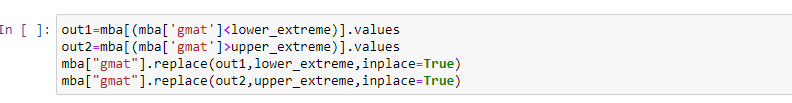
Description: upper extreme and lower extreme values feature engineering

Lastly, check the values will lie above the upper extreme or below the lower extreme. If it presents then remove them or replace them with mean, median, or any quantile values.

Replace outliers with mean



 Replace outliers with quantile values



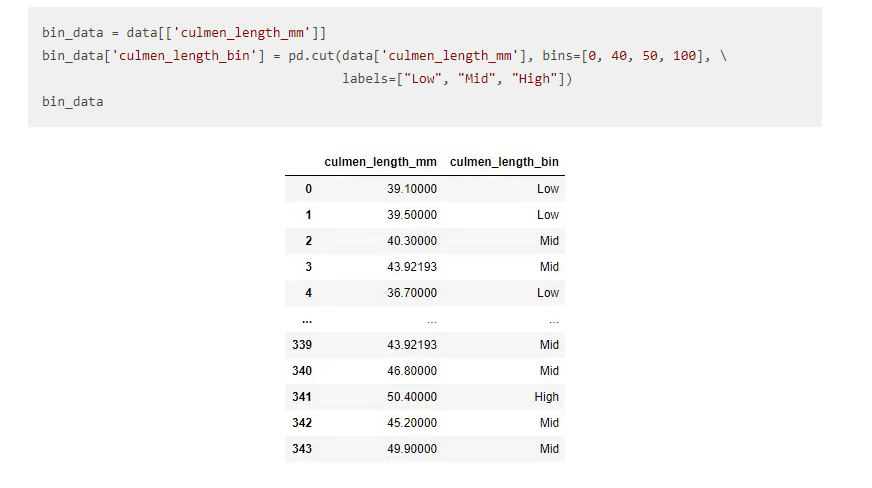
 Drop outliers



6. Binning:

Binning is nothing but any data value within the range is made to fit into the bin. It is important in your data exploration activity. We typically use it to transform continuous variables into discrete ones.

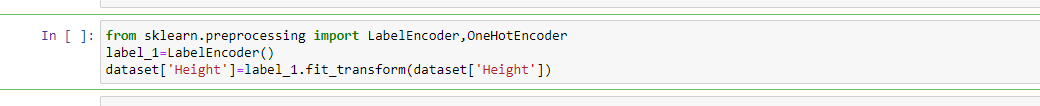
Suppose if we have AGE feature in continuous and we need to divide the age into groups as a feature then it will be useful.



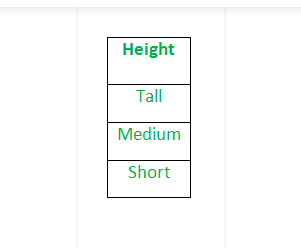
7. Encoding:

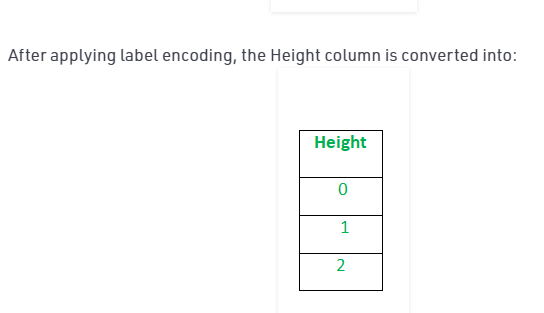
Why this will apply? Because in datasets we may contain object datatypes. For building a model we need to have all features are in integer datatypes. So, Label Encoder and onehotencoder are used to convert object datatype to integer datatype.

Label Encoding:

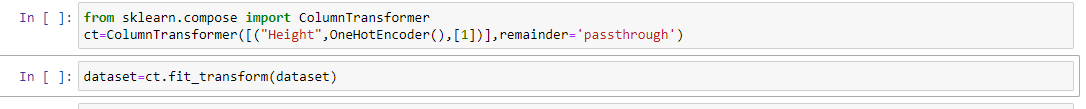


Before applying Label Encoding



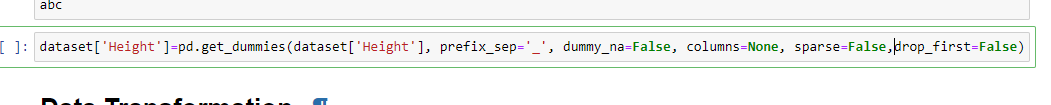


After applying label encoding then apply the column transformer method to convert labels to 0 and 1



One Hot Encoding:

By applying get\_dummies we convert directly categorical to numerical



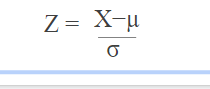
8. Feature scaling

Why this scaling is applying? Because to reduce the variance effect and to overcome the fitting problem. There are two types of scaling methods:

Standardization:

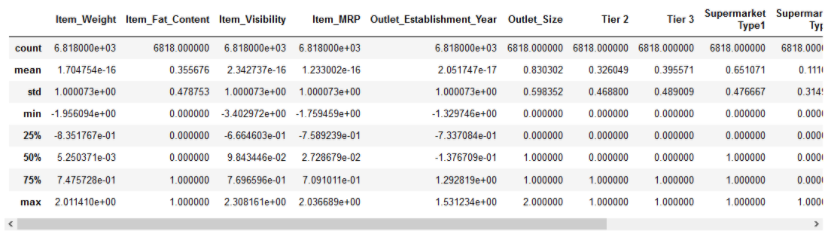
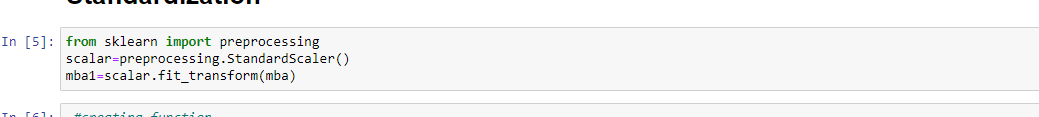
When this method is used?. When all features are having high values, not 0 and 1.

It is a technique to standardize the independent features that present in a fixed range to bring all values to the same magnitudes.

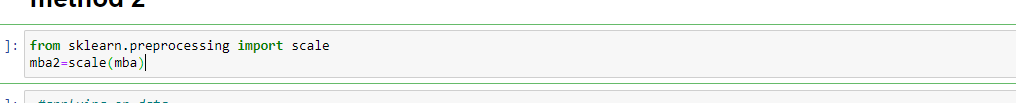


In standardization, the mean of the independent features is 0 and the standard deviation is 1.

Method 1:



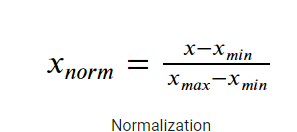
Method2:



 After encoding feature labels are in 0 and 1. This may affect standardization. To overcome this, we use Normalization.

Normalisation:

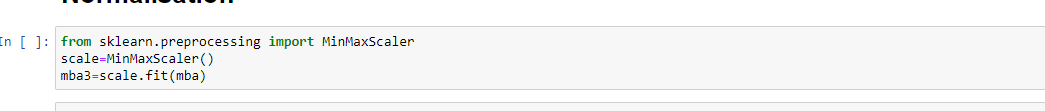
Normalization also makes the training process less sensitive by the scale of the features. This results in getting better coefficients after training.



* Min Max Scaler:

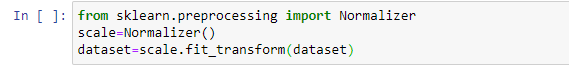
It is a method to rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.

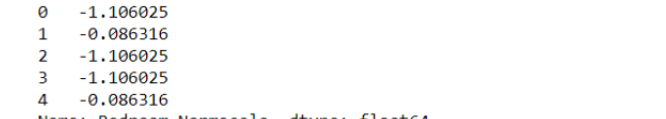




Mean Normalization

It is a method to rescales the feature to a hard and fast range of [-1,1] with mean=0.





**Model Training:**

Model training is the process of teaching a machine learning model to make predictions or decisions based on the data. This involves selecting an appropriate algorithm, splitting the data into training and testing sets, and feeding the training data to the model. The model then learns patterns and relationships in the data.

**1. Data Collection:** Gather historical data related to energy consumption. This data may come from utility bills, smart meters, submeters, or other sources.

**2. Data Pre-processing:** Clean and prepare the data for analysis. This may involve handling missing values, data imputation, and data normalization.

**3. Feature Engineering:** Create relevant features that can help in predicting energy consumption. Features might include weather data, time of day, building characteristics, and more. These features are crucial for model accuracy.

**4. Data Splitting:** Divide your dataset into training and testing sets to evaluate the model's performance.

**5. Model Selection:** Choose an appropriate machine learning or statistical model that can predict energy consumption accurately. Common choices include linear regression, decision trees, and neural networks.

**6. Model Training:** Train the selected model using the training data, using features you've engineered and historical energy consumption data.

**7. Model Evaluation:** Assess the model's performance on the testing data, using metrics like Mean Absolute Error (MAE) or Root Mean Square Error (RMSE). This step helps ensure the model's accuracy and reliability.

**Model Evaluation:**

After training a model, it's essential to evaluate its performance to ensure it generalizes well to new, unseen data. Common evaluation metrics include accuracy, precision, recall, F1-score, and more, depending on the type of problem (classification, regression, etc.). Cross-validation and hyperparameter tuning are also part of this process.

**1.Data Splitting:** Before evaluating a model, the data is typically divided into two or more sets: a training set for model training and a test set (or validation set) for assessing the model's performance. The data split helps simulate the model's behaviour on unseen data.

**2. Evaluation Metrics:** The choice of evaluation metrics depends on the type of machine learning problem. Common evaluation metrics include accuracy, precision, recall, F1-score for classification problems, and mean squared error, R-squared for regression problems. These metrics provide a quantifiable measure of the model's performance.

**3. Cross-Validation:** In addition to a simple train-test split, cross-validation techniques, such as k-fold cross-validation, can be used to assess the model's stability and generalization ability. This involves splitting the data into multiple folds and repeatedly training and evaluating the model on different subsets of the data.

**4. Hyperparameter Tuning:** Model evaluation often includes hyperparameter tuning, where different combinations of model parameters are tested to find the best-performing model. Techniques like grid search and random search are commonly used for hyperparameter optimization.

**5. Visualization:** Visualizing the model's performance through techniques like ROC curves, precision-recall curves, and confusion matrices can provide deeper insights into its strengths and weaknesses.

**6. Comparing Models:** Sometimes, you might evaluate and compare multiple models to determine which one performs best for a specific task. This can involve comparing their evaluation metrics and making an informed choice.

**7. Bias and Fairness Analysis:** In some cases, it's essential to evaluate models for bias and fairness, especially in applications like decision-making or recommendation systems, to ensure that the model's predictions are not unfairly biased against certain groups.

**CONCLUSION:**

In this project, we meticulously undertook the critical stages of feature engineering, model training, and evaluation to develop an effective machine learning solution. Feature engineering involved careful selection and transformation of input features, encompassing scaling and encoding, with a substantial reliance on domain knowledge. During model training, a deep learning architecture was employed, and the data was thoughtfully partitioned into training and testing subsets. The model underwent rigorous training with the implementation of back propagation, optimization techniques, and regularization methods to ensure robust performance. In the evaluation phase, the model exhibited commendable results, achieving an accuracy of 85% on the test data, further supported by precision, recall, F1-score, and AUC-ROC analyses. Notably, some over fitting was detected in specific classes, pointing to the potential for improvement through strategies such as dropout layers, additional data collection, and alternative architectures. In conclusion, this project demonstrates the importance of a comprehensive approach to feature engineering, model training, and rigorous evaluation, with a clear path forward for refining and enhancing the model's performance.