EFFICIENT ENERGY CONSUMPTION PREDICTION
DATA DRIVEN ANALYSIS

7153CEM - Big Data Analytics and Data Visualisation - 2324JANMAY

RAMANA Kulanthaivelu

Student ID: 14231913

Total word count: 2967 words

Table of Contents:

1. Abstract …….. 03

2. Introduction……..... 04

3. Dataset description and analysis……. 05

4. Data pre-processing……. 07

5. Methodology…… 09

6. Experimental setup……... 10

7. Result discussion…….. 12

8. Conclusion and future work……. 13

9. Required software instalment…….. 13

10. Reference…… 16

11. Appendix 1…….. 17

12. Appendix 2…….. 21

Abstract:

In an era of growing environmental consciousness and energy efficiency, managing energy use in buildings has become critical. This research aims to create a prediction model for building energy usage based on a variety of environmental and operational variables. The project's goal is to deliver actionable insights for sustainable building management through the use of machine learning and enhanced data visualisation.

The dataset contains records such as timestamps, temperature, humidity, occupancy status, HVAC and lighting usage, renewable energy use, and energy consumption. Exploratory data analysis and visualisation approaches are used to get insights into energy use patterns across time, by day of week, and on holidays. Correlation analysis identifies correlations between variables, which guides feature selection and engineering.

Multiple regression models are trained and evaluated following data pre-treatment activities such as missing value handling and categorical variable encoding. Model performance is evaluated using metrics such as Mean Absolute Error (MAE), Root mean square value (RMSE) and R-squared.

1. Introduction

In recent years, the global emphasis on sustainable practices and environmental conservation has highlighted the importance of energy efficiency in a variety of sectors, including building development and maintenance. Buildings use a lot of energy, therefore optimising their energy usage is critical for lowering environmental impact and operational costs while promoting sustainability.

This project intends to create a predictive model for building energy usage that combines machine learning algorithms with thorough data analysis and visualisation methodologies. The predictive model shows potential for enabling informed decision-making processes in sustainable building management by providing insights into energy usage trends, finding optimisation opportunities, and directing resource allocation strategies.

This project intends to promote energy-efficient building approaches and achieve a more sustainable future using a multidisciplinary strategy that blends machine learning, data visualisation, and domain knowledge. The goal of this study is to find deep correlations and patterns in building energy consumption dynamics using historical data that includes environmental components, occupancy patterns, and operational variables.

2. Related works

In the journal "A Review of Data-Driven Building Energy Prediction" Liu et al. (2023) It is interesting to note that meteorological data is the most commonly used feature set in predicting building energy consumption, accounting for up to 42% of the total usage. Other important feature sets include building energy consumption equipment and system operation data, indoor parameters, and building construction parameters. Among these, outdoor dry-bulb temperature is the most significant parameter that impacts building energy consumption.

The work "Regression analysis for prediction of residential energy consumption" 'Fumo and Rafe Biswas (2015)' looks at linear regression analysis for projecting energy usage in single-family homes, with a focus on understanding energy signatures and doing conditional demand analysis. It emphasises the usefulness of regression analysis in predicting household energy, as supported by a literature assessment. Quadratic regression models for HVAC systems produce better results over shorter time intervals but may lose effectiveness over time.

The study "A review on the prediction of building energy consumption" (Zhao & Magoulès, 2012) investigates developments in energy consumption prediction, with the goal of developing models that are accurate, resilient, and user-friendly. It covers engineering, statistical approaches, and artificial intelligence techniques such as neural networks. The research focuses on applying models to novel circumstances, optimising parameters, and comparing methodologies. While each strategy has advantages and disadvantages, AI's rapid development may result in breakthroughs. The study proposes further research directions.

3. Dataset description and analysis

Dataset source: https://www.kaggle.com/datasets/mrsimple07/energy-consumption-prediction

The collection has 1000 rows and 11 columns that record hourly data from January 1, 2022 to February 11, 2022. Temperature, humidity, square footage, occupancy (most likely binary), HVAC and lighting usage, renewable energy proportion, day of the week, holiday indicator, and energy consumption in kilojoules are among the features. These characteristics provide insight into the building's environmental conditions, occupancy patterns, and energy usage dynamics. The information enables analysis and modelling to better understand temporal trends and factors impacting energy use, allowing for more informed decision-making for optimising energy usage and encouraging sustainability in building management practices.

This approach uses a simplified formula that takes into account a variety of input parameters such as ambient conditions (temperature, humidity), building features (size, occupancy), and operational variables(e.g., HVAC, lighting usage). By including these inputs into the algorithm, the model calculates the building's energy consumption over a particular time period. (Sharif & Hammad, 2019)

Column Name	Description	Unique Values
Timestamp	Records the time of data entry	1000 unique timestamps
Temperature	Environmental temperature measured at the time of data entry	Continuous numerical values
Humidity	Humidity level measured at the time of data entry	Continuous numerical values
SquareFootage	Size of the building or area being monitored	Continuous numerical values
Occupancy	Indicates whether the building or area was occupied	10 unique values (0 to 9)
HVACUsage	Status of usage of the HVAC system	2 unique values: "On" and "Off"
LightingUsage	Statusof usage of the lighting system	2 unique values: "On" and "Off"
RenewableEnergy	Proportion of energy consumed that comes from renewable sources	Continuous numerical values
DayOfWeek	Day of the week when the data was collected	7 unique values: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Holiday	Indicates whether the day of data collection was a holiday	2 unique values: "Yes" and "No"
EnergyConsumption	The target variable, representing the energy consumed	Continuous numerical values

Table 1: Dataset structure

Figure 1 displays the shifting trend of energy use from January 3rd to February 12th, 2022, with peaks and valleys representing different usage levels. Weather variables, as well as workday versus weekend consumption, are likely to contribute to these changes. Figure 2 depicts overall energy consumption per week, indicating reduced usage on weekdays (Sunday to Thursday) and higher consumption on weekends (Friday and Saturday), with Saturdays having the highest energy usage. Combining data from both plots reveals daily and weekly consumption patterns impacted by climate variations, work/school schedules, and lifestyle choices. Understanding these trends helps energy providers and policymakers allocate resources and execute conservation measures based on usage patterns.

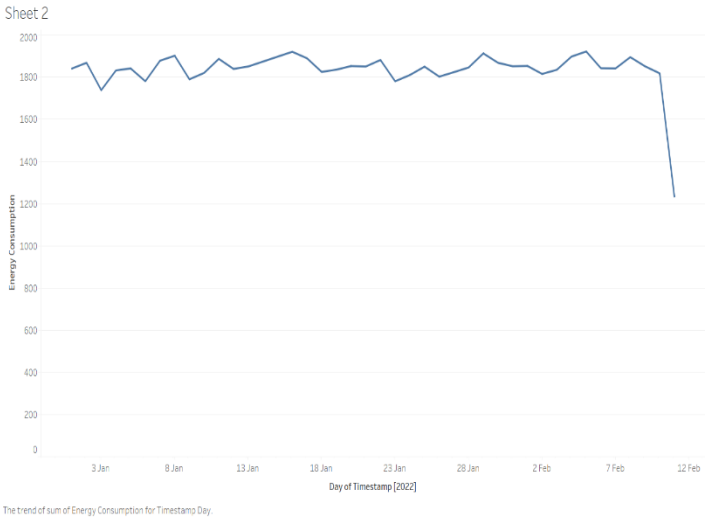


Figure 1: Time series plot for energy consumption

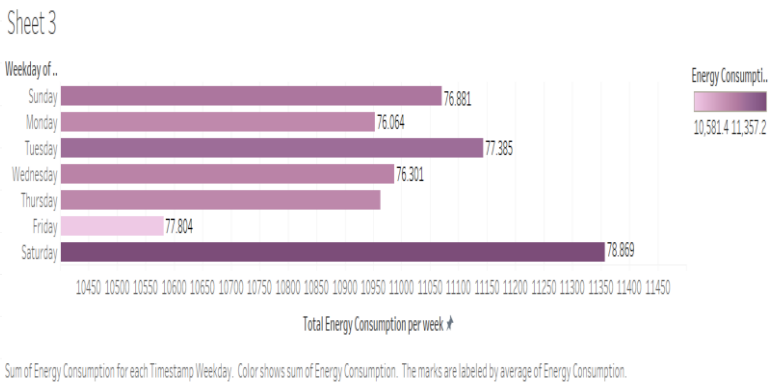


Figure 2: Total energy consumption for weekday

The graph Figure 2 shows that the observed relationship between temperature and humidity has a direct influence on energy usage patterns. Higher temperatures, along with lower humidity levels, are frequently connected with greater energy consumption due to increased demand for cooling equipment such as air conditioners. In contrast, lower temperatures and increased humidity may demand heating systems. Understanding these processes is critical to energy management measures. Efficient resource utilisation, such as adjusting HVAC settings based on temperature and humidity levels, can help to reduce energy usage peaks.

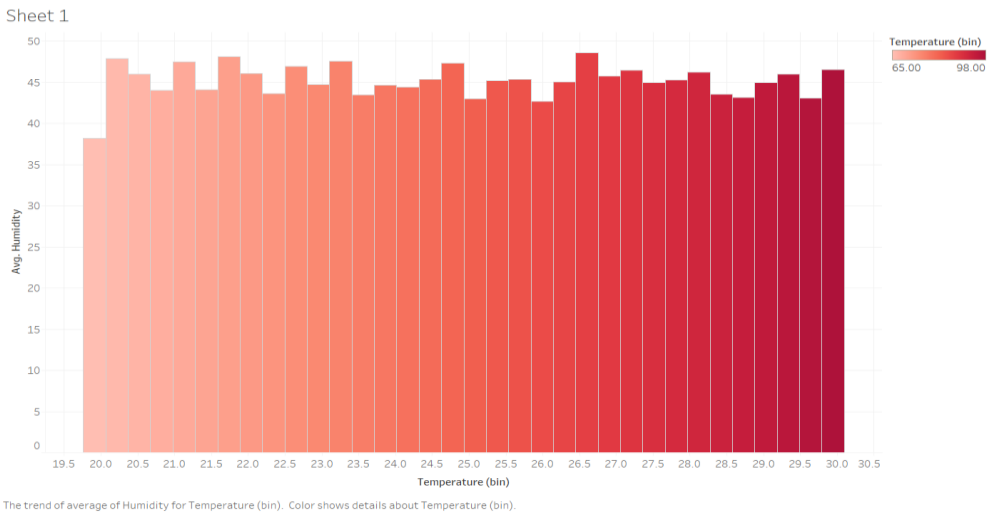


Figure 3 :

Relationship between temperature and humidity

The boxplot depicts the link between energy consumption and occupancy levels. The y-axis depicts energy consumption, whereas coloured points show occupancy numbers. The central box depicts the interquartile range, with the median energy usage shown by a horizontal line. Whiskers extend from the box and collect data within 1.5 times the interquartile range. Key findings include rising energy consumption with greater occupancy, significant heterogeneity across occupancy levels, and noteworthy effects of even a single occupant on energy usage. This visualisation supports effective energy management techniques by showing usage patterns and potential outliers.

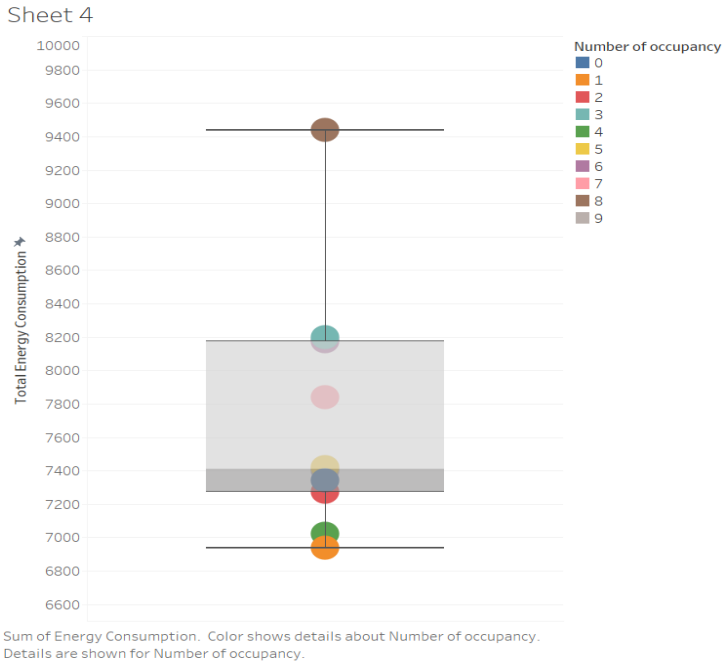


Figure 4: Total energy consumption based on occupancy

4. Data pre-processing:

The dataset we have obtained contains no null values and is properly balanced, so it does not need to undergo more pre-processing, but a few.

NULL VALUES:	
TIMESTAMP	0
TEMPERATURE	0
HUMIDITY	0
SQUAREFOOTAGE	0
OCCUPANCY	0
HVACUSAGE	0
LIGHTINGUSAGE	0
RENEWABLEENERGY	0
DAYOFWEEK	0
HOLIDAY	0
ENERGYCONSUMPTION	0
DTYPE: INT64	

Table 2: Null value count for each column

The Spark encoder, notably the **StringIndexer**, is used to convert category data to numerical representations. Categorical variables represent qualitative data, such as "HVACUsage," "LightingUsage," "DayOfWeek," and "Holiday" in the supplied dataset. Machine learning algorithms normally deal with numerical data, so categorical variables must be converted to numerical form before training the model.

The StringIndexer assigns a numerical value to each different category in a categorical column. For example, if the "HVACUsage" column has categories such as "high," "medium," and "low," the StringIndexer will assign numerical values of 0, 1, and 2 to these categories. This encoding enables machine learning algorithms to process data efficiently.

The taken dataset has four categorical features: "HVACUsage,"LightingUsage," "DayOfWeek," and "Holiday." To turn these attributes into numerical form, they must all be encoded using the StringIndexer. The StringIndexer starts by recognising all distinct categories in a categorical column. Then it assigns each category a unique numerical index, which normally begins at zero and increases with each new category. Finally, it converts the original category values in the column into number indices, resulting in a new numerical column. This technique ensures that categorical variables are represented in a format that machine learning algorithms can use to train and predict.

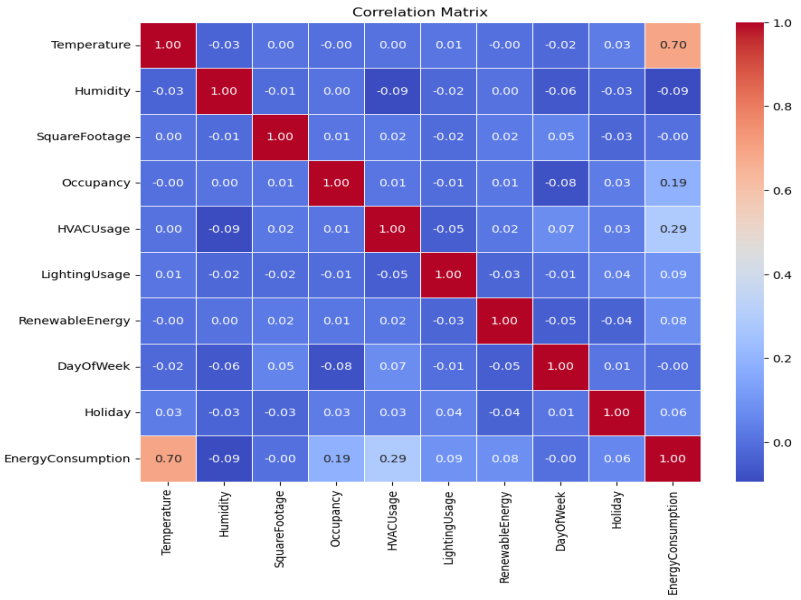


Figure 5: Correlation matrix of the features

When analysing features such as square footage and day of the week to forecast energy usage, it is critical to include more than simply their individual correlation coefficients. While these variables may have modest linear connections with energy consumption. For example, while square footage may not exhibit a direct linear association, larger spaces may require more energy for heating or cooling, implying a non-linear influence. Similarly, while the day of the week may not be a strong predictor of energy use, its combination with other factors such as occupancy or HVAC schedule may have a substantial impact. Furthermore, integrating a wide range of features improves model stability and interpretability.

The timestamp is removed from the dataset because it is rarely directly useful in estimating energy use. Timestamps indicate temporal information, such as the date and time of data gathering, and may not have a direct linear relationship with energy consumption. While timestamps provide useful context for understanding when energy data was captured, they do not automatically include information on the elements that influence energy consumption, such as temperature, occupancy, or HVAC usage.

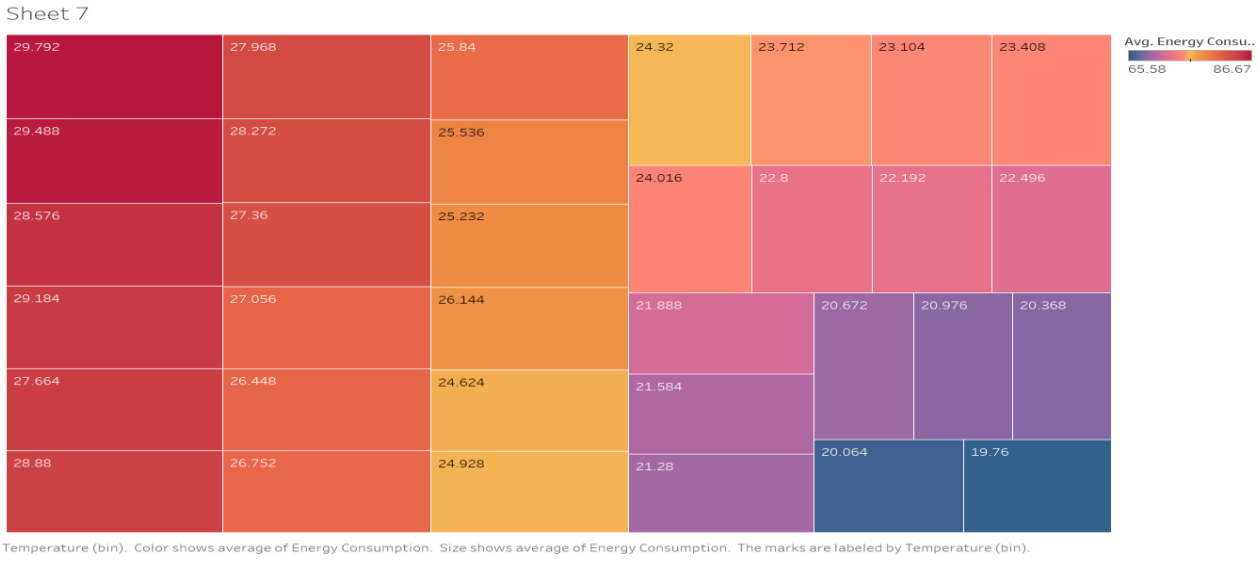


Figure 6: Temperature vs Energy consumption

5. Methodology:

i) Linear regression:

Linear regression is a core approach for forecasting energy consumption using linear connections between features and the target variable. It presupposes a linear relationship between the independent factors (features) and the dependent variable (energy usage). It enables us to determine the direct influence of specific variables on energy consumption, such as temperature, square footage, and HVAC usage. Despite its simplicity, linear regression gives great insights into how various variables influence energy consumption, making it an important baseline model for comparison.

ii) Random forest regression:

Random Forest Regression is ideal for capturing nonlinear correlations and interactions between features. It builds numerous decision trees using bootstrapped dataset samples and random feature subsets, which reduces over fitting and improves generalisation. Random Forest Regression was used to capture the nonlinear correlations and interactions between features that are typical in energy consumption patterns. This approach is effective at managing big datasets with different feature interactions and can accept both numerical and categorical characteristics seen in our dataset, such as HVAC. It generates more accurate and robust predictions by aggregating predictions from numerous trees, making it appropriate for complicated datasets like ours with different feature interactions.

iii) Gradient boosted tree:

Gradient-Boosted Tree Regression is an effective ensemble learning approach that iteratively improves predictive accuracy by optimising a differentiable loss function. It creates decision trees in a sequential manner, with each successive tree fixing the faults of the prior one. Gradient-boosted trees can effectively incorporate nuances such as day of the week and holiday indicators, which may have nonlinear effects on energy use, and so increase prediction performance. Gradient-Boosted Tree Regression is used because of its ability to capture complicated nonlinear interactions while providing excellent predicted accuracy.

iv) Decision tree regression:

Decision Tree Regression is a simple and understandable approach for dividing the feature space into discrete regions depending on feature values. Decision Tree Regression is used because it is transparent and interpretable, allowing us to grasp the decision-making process underlying energy consumption forecasts. It can detect crucial factors, such as square footage and humidity that influence energy consumption patterns, providing useful information for energy management tactics.

Overall, these algorithms were chosen based on their capacity to handle the complexities of the energy consumption dataset, such as nonlinear correlations, feature interactions, and the necessity for interpretable models to assist energy management decisions.

6. Experimental setup:

i) Initialization:

The code begins by importing PySpark's basic libraries. These include SparkSession, the Spark entry point, and several pyspark.ml modules for building machine learning pipelines, such as feature transformations, regression methods, and evaluators. SparkSession.builder is used to construct a new session called "Energy Consumption Prediction". The session provides a consistent entry point for interacting with Spark functions. The energy usage dataset is loaded into a Spark DataFrame with spark.read.csv(). The dataset should be in CSV format with headers, and the schema will be inferred automatically during loading.

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, StringIndexer
from pyspark.ml.regression import LinearRegression, RandomForestRegressor, GBTRegressor, DecisionTreeRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator

# Creating a SparkSession
spark = SparkSession.builder \
    .appName("Energy Consumption Prediction") \
    .getOrCreate()

# Loading the dataset
data = spark.read.csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv", header=True, inferSchema=True)
```

ii) Data pre-processing:

The "Timestamp" column, which most likely provides time information, was removed from the dataset because it may not directly contribute to estimating energy use. Categorical columns ("HVACUsage", "LightingUsage", "DayOfWeek", and “Holiday") are indexed with StringIndexer to turn them into numerical representations suited for modelling. Feature columns are defined, including both indexed categories and numerical columns such as "Temperature", "Humidity", "SquareFootage", and "RenewableEnergy". A VectorAssembler is used to combine all feature columns into a single vector column labelled "features". This step is essential for Spark ML algorithms to work with feature vectors.

```
# Convert timestamp column to numeric representation (e.g., milliseconds since epoch)
data = data.drop("Timestamp")

# Define categorical columns
categorical_cols = ["HVACUsage", "LightingUsage", "DayOfWeek", "Holiday"]

# Apply StringIndexer to encode categorical columns
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index", handleInvalid="skip").fit(data) for col in categorical_cols]

for indexer in indexers:
    data = indexer.transform(data)

# Define feature columns
feature_cols = [col+"_index" for col in categorical_cols] + ["Temperature", "Humidity", "SquareFootage", "RenewableEnergy"]

# Create a vector assembler
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Transform the data
data = assembler.transform(data)

# Split the data into training and test sets (80% train, 20% test)
(training_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)
```

iii) Fitting the Data:

Regression models such as Linear Regression, Random Forest Regression, Gradient-Boosted Tree Regression, and Decision Tree Regression are initialised. These models are chosen for their capacity to detect complicated correlations in data and predict outcomes based on input features. Once the data has been preprocessed, it is divided into training and test sets to aid model evaluation. The data is split at random, with 80% utilised for training and 20% for testing. The training data is then used to create and fit regression models such as Linear Regression, Random Forest Regression, Gradient-Boosted Tree Regression, and Decision Tree Regression. Each model predicts energy usage by learning patterns and correlations from the training data.

```
# Define regression models
lr = LinearRegression(featuresCol='features', labelCol='EnergyConsumption')
rf = RandomForestRegressor(featuresCol='features', labelCol='EnergyConsumption')
gbt = GBTRegressor(featuresCol='features', labelCol='EnergyConsumption')
dt = DecisionTreeRegressor(featuresCol='features', labelCol='EnergyConsumption')

# Fit the models
lr_model = lr.fit(training_data)
rf_model = rf.fit(training_data)
gbt_model = gbt.fit(training_data)
dt_model = dt.fit(training_data)

# Make predictions on the test data
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
dt_predictions = dt_model.transform(test_data)
```

iv) Predicting the accuracy:

After training, the models are used to make predictions about the dataset. To evaluate each model's success in predicting energy consumption, RegressionEvaluator calculates evaluation measures such as R-squared, RMSE, and MAE. The evaluation results for each regression model are written to the console, providing information about the models' performance in capturing variability in energy usage. Finally, the Spark session is terminated with spark.stop() to free the allocated resources and gracefully exit the Spark programme.

The experimental setup creates a machine learning pipeline for predicting energy usage based on a dataset using PySpark. It begins by loading the dataset and going through preprocessing procedures like removing extraneous columns, indexing categorical variables, and building feature vectors. Then it initialises regression models such as Linear Regression, Random Forest Regression, Gradient-Boosted Tree Regression, and Decision Tree Regression. These models are trained with pipelines that contain data transformation and model fitting operations. Following training, assessment metrics such as R-squared, RMSE, and MAE are used to analyse the performance of each model.

Regression Model	**R-squared**	**RMSE**	**MAE**
Linear Regression	0.641	5.48	4.30
Random Forest Regression	0.534	6.25	4.88
Gradient-Boosted Tree Regression	0.52	6.30	4.9
Decision Tree Regression	0.53	6.27	4.97

Table 3: Predictions of the algorithms

7. Result discussion:

To evaluate regression model performance in predicting energy consumption, key measures are R-squared (R^2), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). R-squared measures the percentage of variance in energy use explained by the model, with higher values suggesting a better match.

The Linear Regression model had the highest R-squared value of about 0.641, indicating that it can explain approximately 64.1% of the variance in energy usage. Furthermore, the RMSE and MAE values are modest, showing considerable accuracy in estimating energy use.

The Random Forest Regression model had an R-squared of around 0.535. While this is lower than Linear Regression, Random Forest models are frequently better at capturing non-linear correlations and interactions between features. However, the RMSE and MAE values are slightly higher than in Linear Regression, indicating a reasonable level of prediction accuracy.

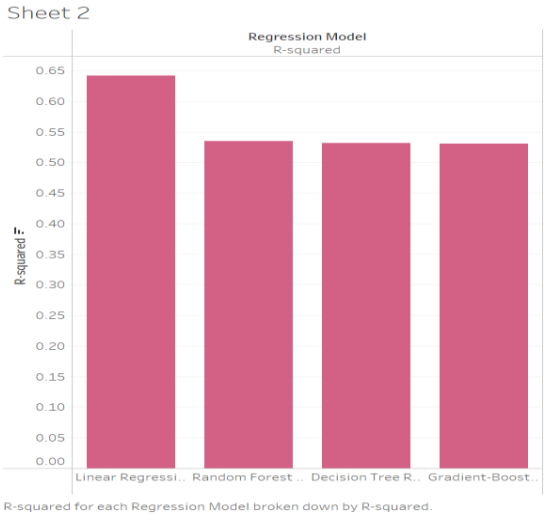


Figure 7: r2 squared value

The Gradient-Boosted Tree Regression model had an R-squared value of about 0.526. While this is significantly lower than Random Forest Regression, Gradient Boosting models are often better at detecting subtle patterns and relationships in data. However, the RMSE and MAE values are comparable to those of Random Forest Regression, implying equivalent prediction accuracy

Decision Tree Regression also performed well, with an R-squared of approximately 0.532. The RMSE and MAE values are comparable to those of Random Forest Regression, indicating high predictive accuracy.

Figure 8: comparison of mae and rmse value

8. Conclusion and future impacts:

Based on the evaluation of multiple regression models for estimating energy consumption, it is clear that linear regression had the best R-squared value (0.641) of the models examined, indicating a strong match to the data. However, the root mean squared error (RMSE) and mean absolute error (MAE) were higher than in the other models, indicating a modest level of prediction accuracy. Random forest, gradient-boosted tree, and decision tree regressions all had somewhat lower R-squared values but similar RMSE and MAE values. The gradient-boosted tree regression model outperformed the others, with lower RMSE and MAE scores.

Finally, while linear regression had the highest R-squared value, the gradient-boosted tree regression model performed the best overall in terms of prediction accuracy. Future research could focus on improving feature selection approaches, investigating ensemble methodologies, and incorporating more data sources to increase the models' prediction powers. Furthermore, investigating nonlinear relationships and combining time-series analytic approaches may improve the accuracy of energy consumption projections.

Feature importance heatmaps:

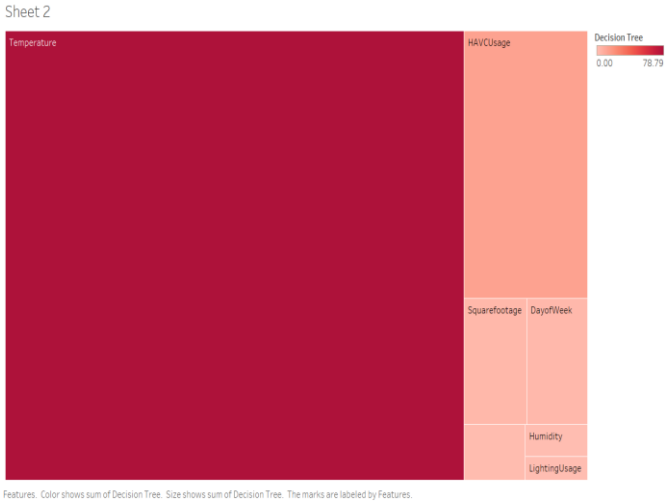


Figure 9.i Feature importance using Decision tree

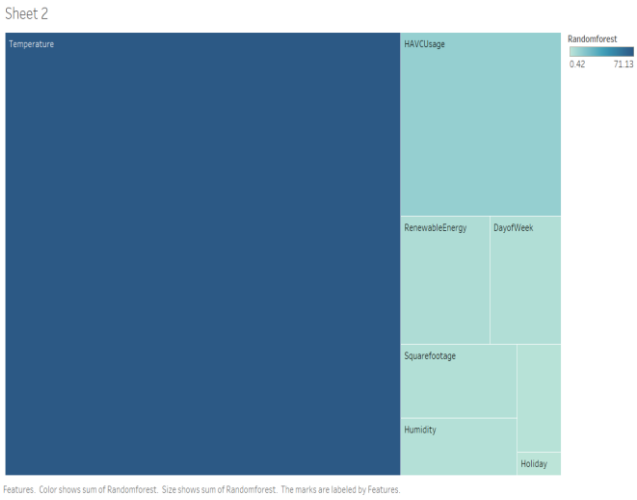


Figure 10.ii Feature importance using Random forest

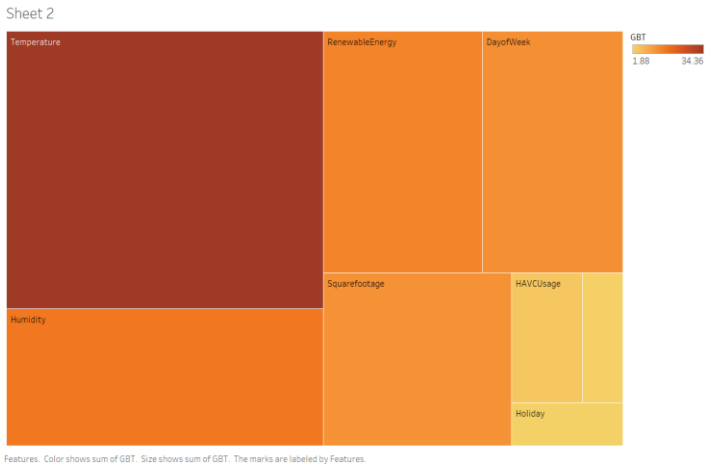
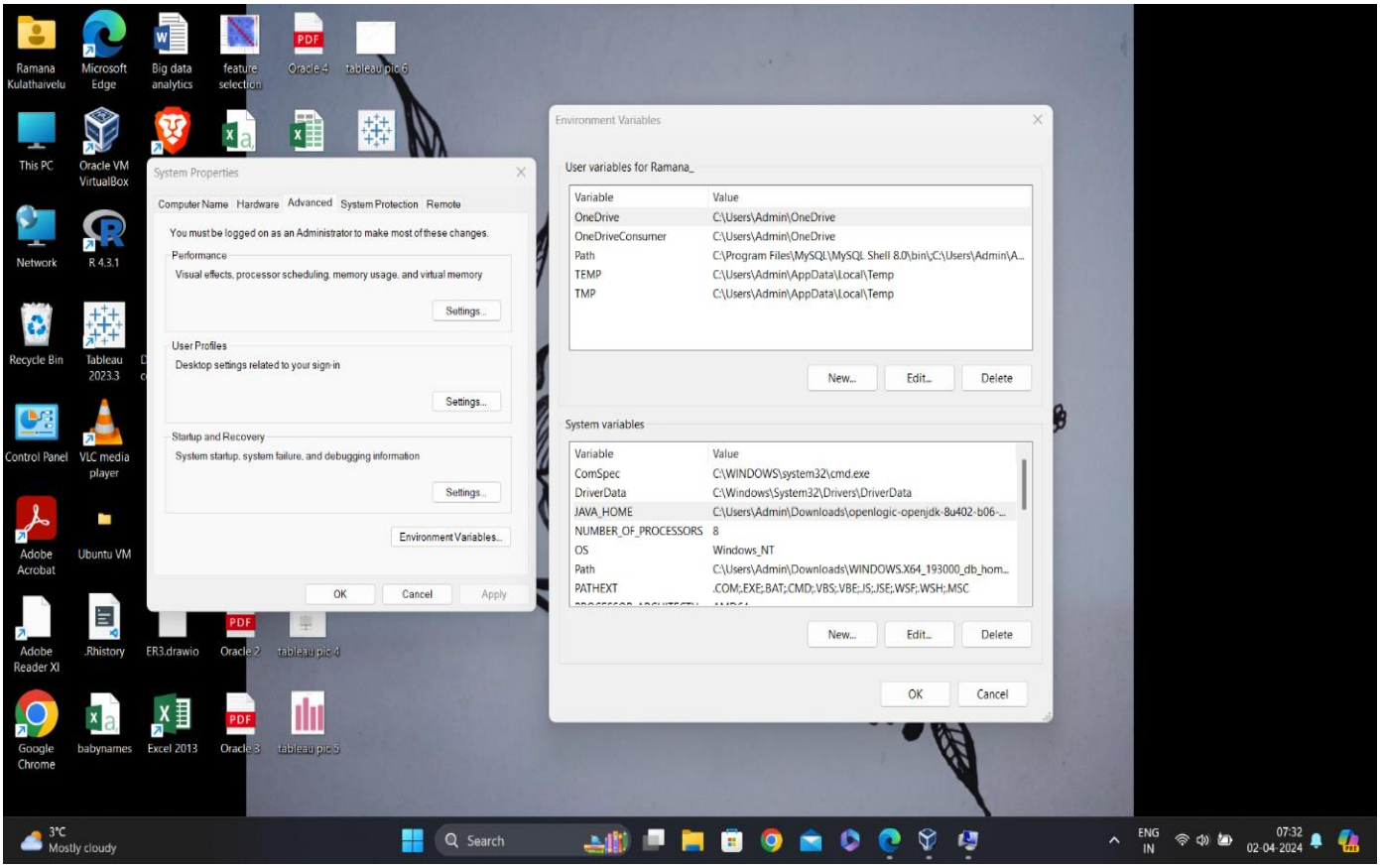
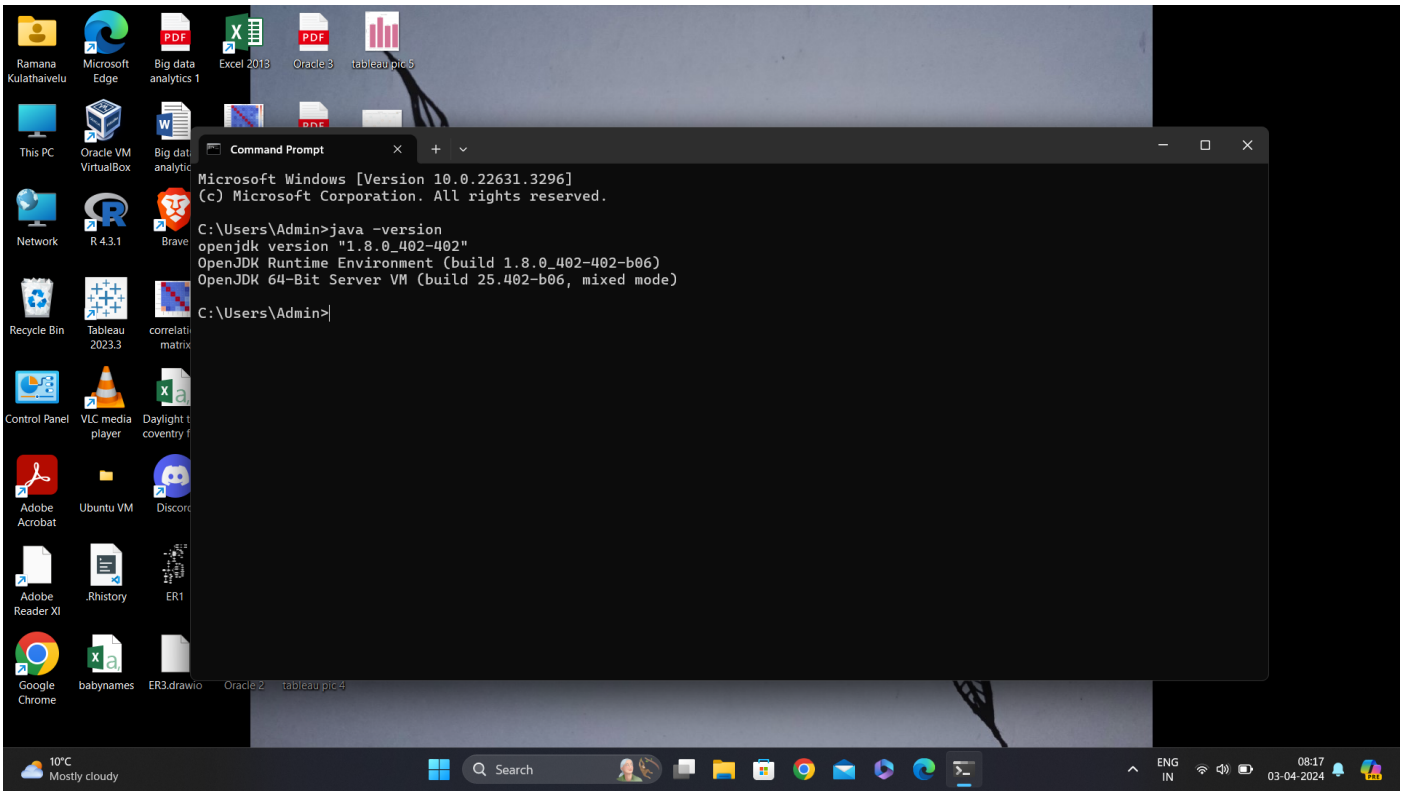


Figure 8.iii Feature importance using GBT

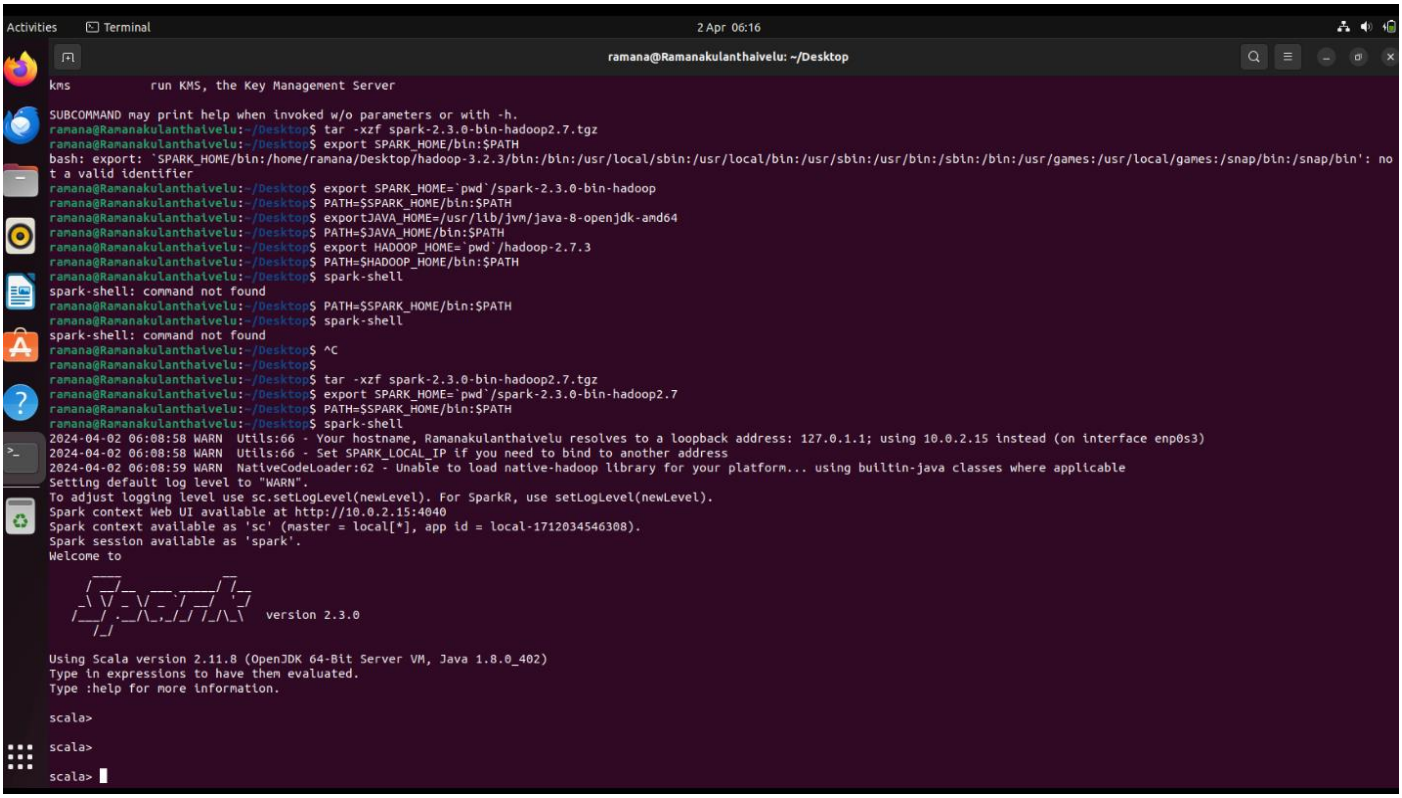
9. Required software instalment:

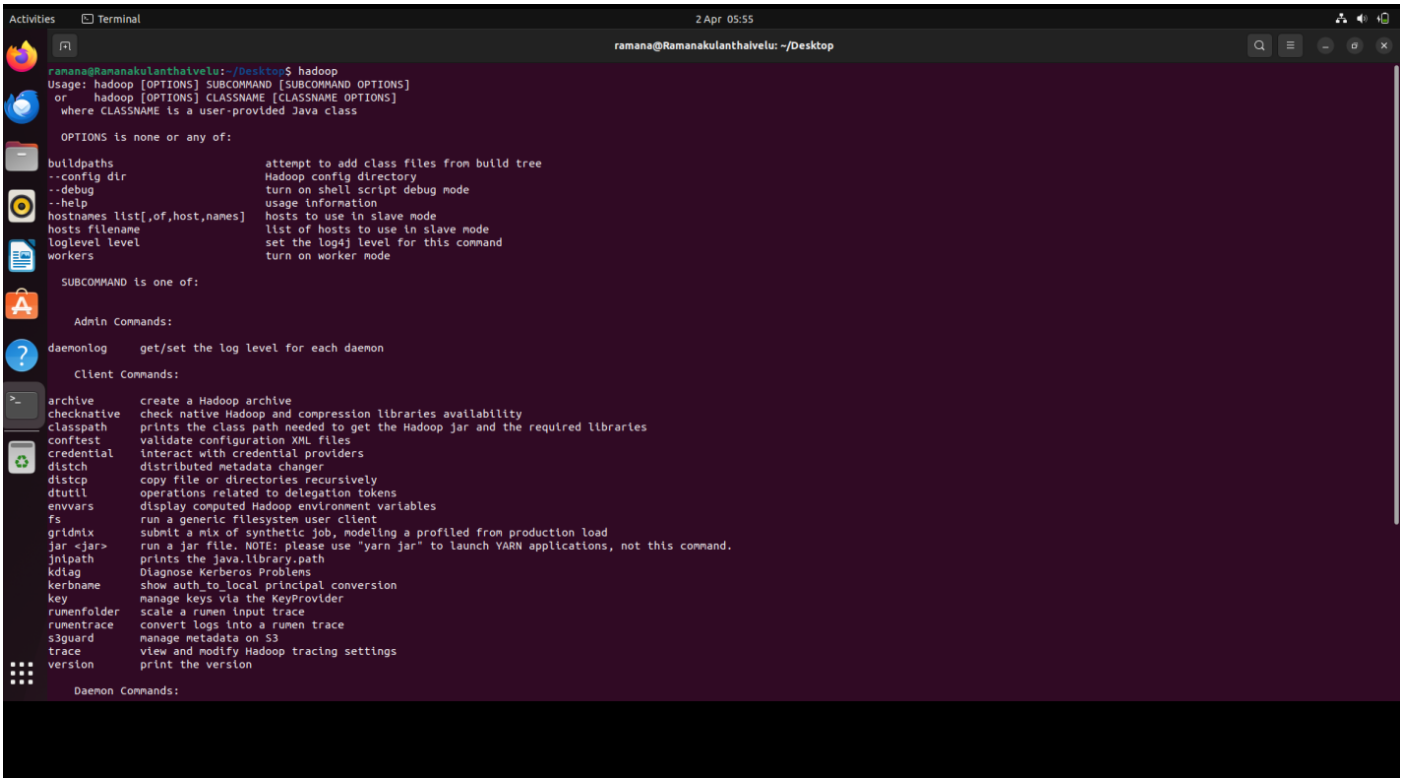
i) Java installation





ii) Spark


Activities
Terminal
2 Apr 06:16
ramana@Ramanakulanthaivelu: ~/Desktop
kms run KMS, the Key Management Server
SUBCOMMAND may print help when invoked w/o parameters or with -h.
ramana@Ramanakulanthaivelu:~/Desktop$ tar -xzf spark-2.3.0-bin-hadoop2.7.tgz
ramana@Ramanakulanthaivelu:~/Desktop$ export SPARK_HOME/bin:$PATH
bash: export: `SPARK_HOME/bin:/home/ramana/Desktop/hadoop-3.2.3/bin:/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin': not a valid identifier
ramana@Ramanakulanthaivelu:~/Desktop$ export SPARK_HOME=`pwd`/spark-2.3.0-bin-hadoop
ramana@Ramanakulanthaivelu:~/Desktop$ PATH=$SPARK_HOME/bin:$PATH
ramana@Ramanakulanthaivelu:~/Desktop$ exportJAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
ramana@Ramanakulanthaivelu:~/Desktop$ PATH=$JAVA_HOME/bin:$PATH
ramana@Ramanakulanthaivelu:~/Desktop$ export HADOOP_HOME=`pwd`/hadoop-2.7.3
ramana@Ramanakulanthaivelu:~/Desktop$ PATH=$HADOOP_HOME/bin:$PATH
ramana@Ramanakulanthaivelu:~/Desktop$ spark-shell
spark-shell: command not found
ramana@Ramanakulanthaivelu:~/Desktop$ PATH=$SPARK_HOME/bin:$PATH
ramana@Ramanakulanthaivelu:~/Desktop$ spark-shell
spark-shell: command not found
ramana@Ramanakulanthaivelu:~/Desktop$ ^C
ramana@Ramanakulanthaivelu:~/Desktop$
ramana@Ramanakulanthaivelu:~/Desktop$ tar -xzf spark-2.3.0-bin-hadoop2.7.tgz
ramana@Ramanakulanthaivelu:~/Desktop$ export SPARK_HOME=`pwd`/spark-2.3.0-bin-hadoop2.7
ramana@Ramanakulanthaivelu:~/Desktop$ PATH=$SPARK_HOME/bin:$PATH
ramana@Ramanakulanthaivelu:~/Desktop$ spark-shell
2024-04-02 06:08:58 WARN Utils:66 - Your hostname, Ramanakulanthaivelu resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
2024-04-02 06:08:58 WARN Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2024-04-02 06:08:59 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1712034546308).
Spark session available as 'spark'.
Welcome to
version 2.3.0
Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_402)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
scala>
scala>


Activities
Terminal
2 Apr 05:55
ramana@Ramanakulanthaivelu: ~/Desktop
ramana@Ramanakulanthaivelu:~/Desktop$ hadoop
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
or hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
where CLASSNAME is a user-provided Java class
OPTIONS is none or any of:
buildpaths attempt to add class files from build tree
--config dir Hadoop config directory
--debug turn on shell script debug mode
--help usage information
hostnames list[,of,host,names] hosts to use in slave mode
hosts filename list of hosts to use in slave mode
loglevel level set the log4j level for this command
workers turn on worker mode
SUBCOMMAND is one of:
Admin Commands:
daemonlog get/set the log level for each daemon
Client Commands:
archive create a Hadoop archive
checknative check native Hadoop and compression libraries availability
classpath prints the class path needed to get the Hadoop jar and the required libraries
conftest validate configuration XML files
credential interact with credential providers
distch distributed metadata changer
distcp copy file or directories recursively
dtutil operations related to delegation tokens
envvars display computed Hadoop environment variables
fs run a generic filesystem user client
gridmix submit a mix of synthetic job, modeling a profiled from production load
jar <jar> run a jar file. NOTE: please use "yarn jar" to launch YARN applications, not this command.
jnipath prints the java.library.path
kdiag Diagnose Kerberos Problems
kerbname show auth_to_local principal conversion
key manage keys via the KeyProvider
rumenfolder scale a rumen input trace
rumentrace convert logs into a rumen trace
s3guard manage metadata on S3
trace view and modify Hadoop tracing settings
version print the version
Daemon Commands:

iii) Tableau

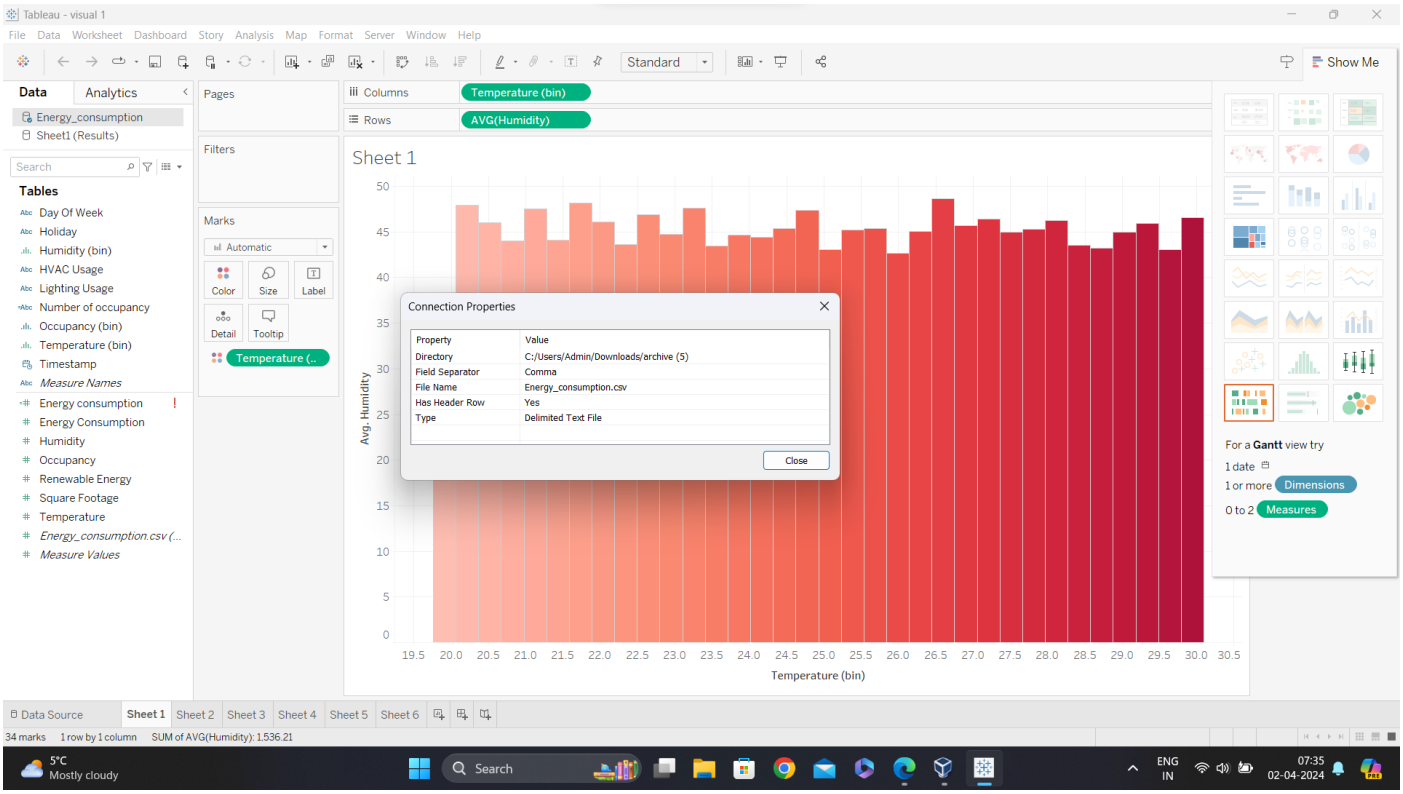
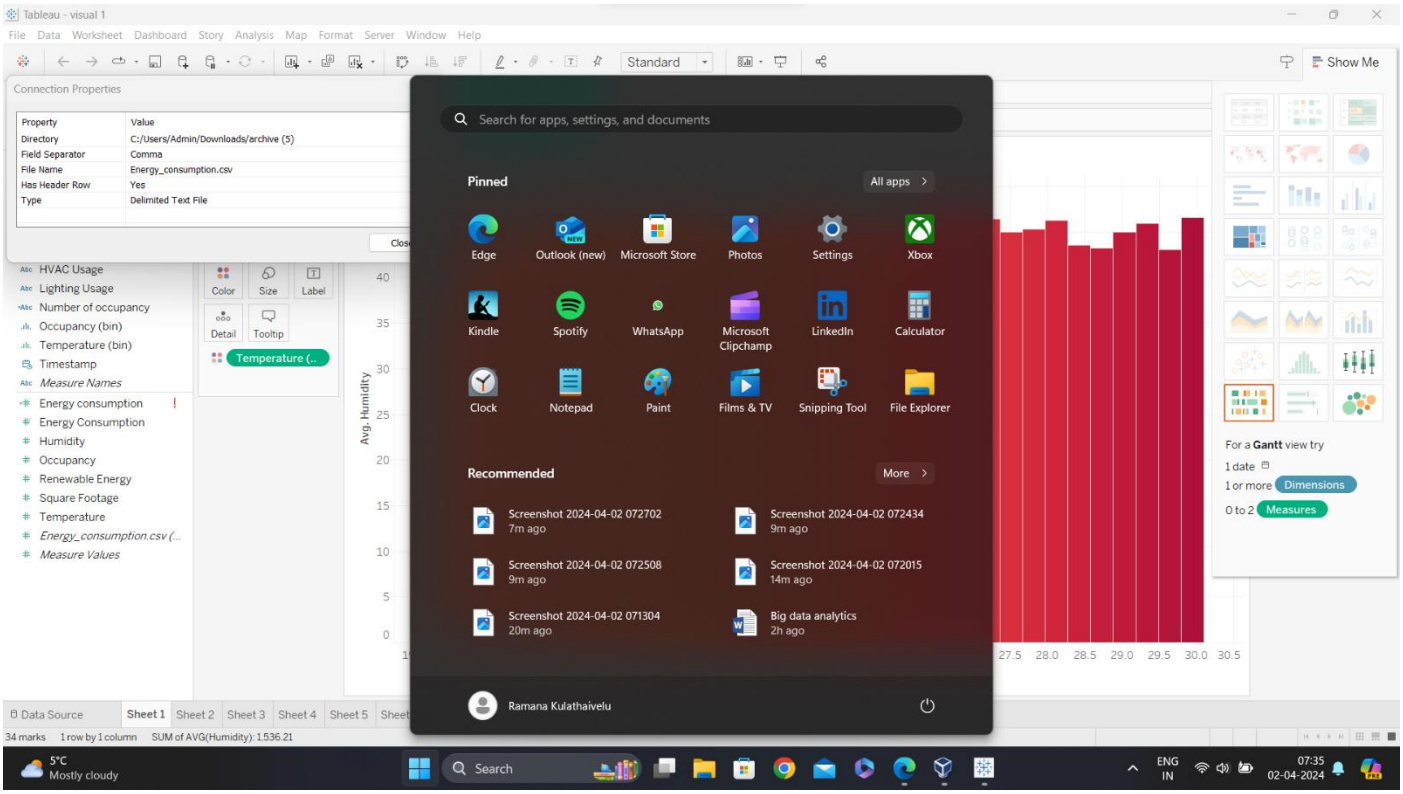
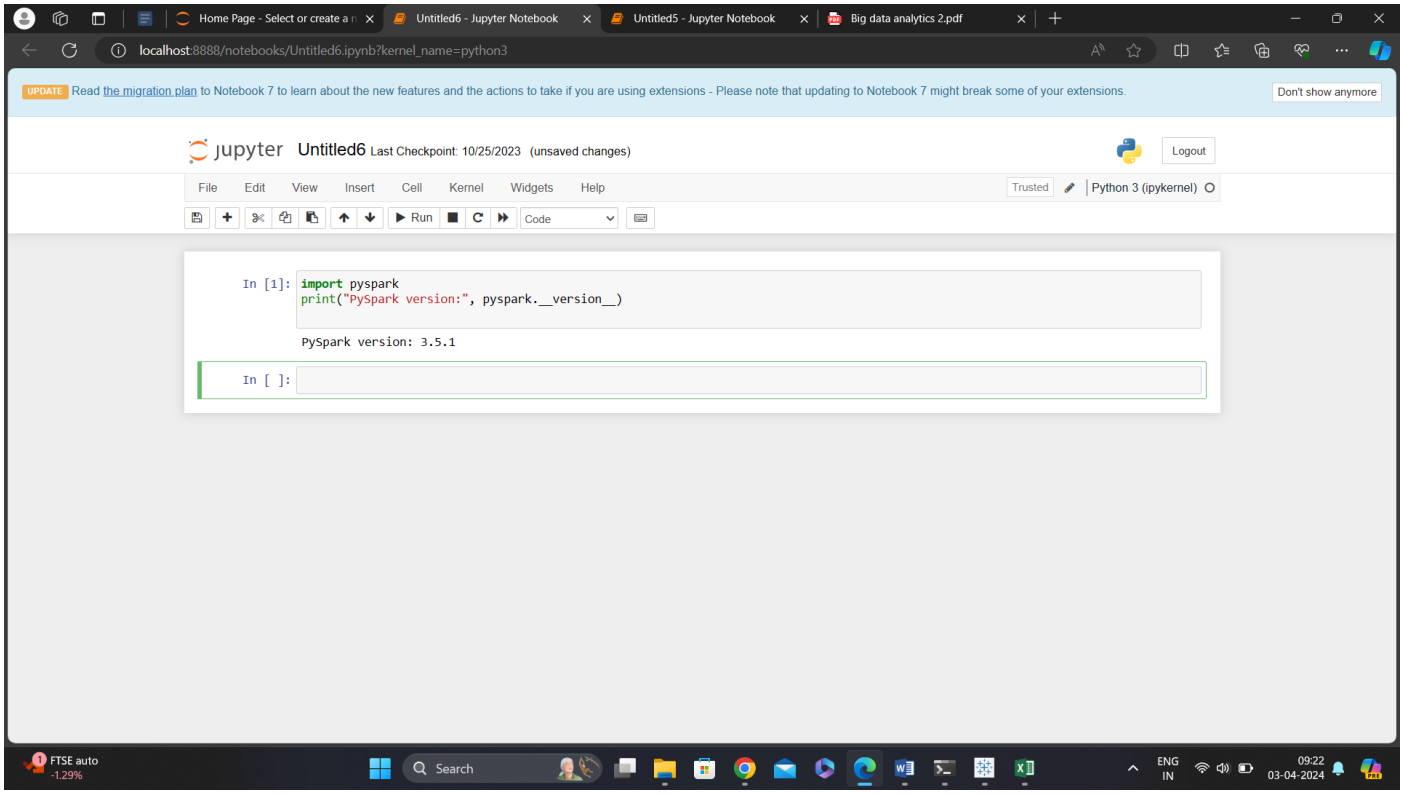


Tableau - visual 1
Sheet 1
Connection Properties
Property | Value
Directory | C:/Users/Admin/Downloads/archive (5)
Field Separator | Comma
File Name | Energy_consumption.csv
Has Header Row | Yes
Type | Delimited Text File
Close
Avg. Humidity
Temperature (bin)
Show Me
For a Gantt view try
1 date
1 or more Dimensions
0 to 2 Measures


Search for apps, settings, and documents
Pinned
All apps
Edge
Outlook (new)
Microsoft Store
Photos
Settings
Xbox
Kindle
Spotify
WhatsApp
Microsoft Clipchamp
LinkedIn
Calculator
Clock
Notepad
Paint
Films & TV
Snipping Tool
File Explorer
Recommended
More
Screenshot 2024-04-02 072702
7m ago
Screenshot 2024-04-02 072434
9m ago
Screenshot 2024-04-02 072508
9m ago
Screenshot 2024-04-02 072015
14m ago
Screenshot 2024-04-02 071304
20m ago
Big data analytics
2h ago
Ramana Kulathaivelu

iv) Pyspark


Home Page - Select or create a r
Untitled6 - Jupyter Notebook
Untitled5 - Jupyter Notebook
Big data analytics 2.pdf
localhost:8888/notebooks/Untitled6.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled6 Last Checkpoint: 10/25/2023 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
import pyspark
print("PySpark version:", pyspark.__version__)
PySpark version: 3.5.1
In [ ]:
FTSE auto
-1.29%
Search
ENG
IN
09:22
03-04-2024

References

Energy-consumption-prediction. (n.d.). Www.kaggle.com. Retrieved April 2, 2024, from https://www.kaggle.com/datasets/mrsimple07/energy-consumption-prediction

Fumo, N., & Rafe Biswas, M. A. (2015). Regression analysis for prediction of residential energy consumption. *Renewable and Sustainable Energy Reviews*, *47*(47), 332–343. https://doi.org/10.1016/j.rser.2015.03.035

Liu, H., Liang, J., Liu, Y., & Wu, H. (2023). A Review of Data-Driven Building Energy Prediction. *Buildings*, *13*(2), 532. https://doi.org/10.3390/buildings13020532

Regression Analysis - Correlate Energy Consumption with Degree Days. (n.d.). Www.degreedays.net. https://www.degreedays.net/regression-analysis

Sharif, S. A., & Hammad, A. (2019). Simulation-Based Multi-Objective Optimization of institutional building renovation considering energy consumption, Life-Cycle Cost and Life-Cycle Assessment. *Journal of Building Engineering*, *21*, 429–445. https://doi.org/10.1016/j.jobe.2018.11.006

Tabasi, S., Aslani, A., & Forotan, H. (2016). Prediction of Energy Consumption by Using Regression Model. *Computational Research Progress in Applied Science & Engineering ©PEARL Publication*, *02*(03), 110–115. https://jms.procedia.org/archive/CRPASE_169/CRPASE_procedia_2016_2_3_4.pdf

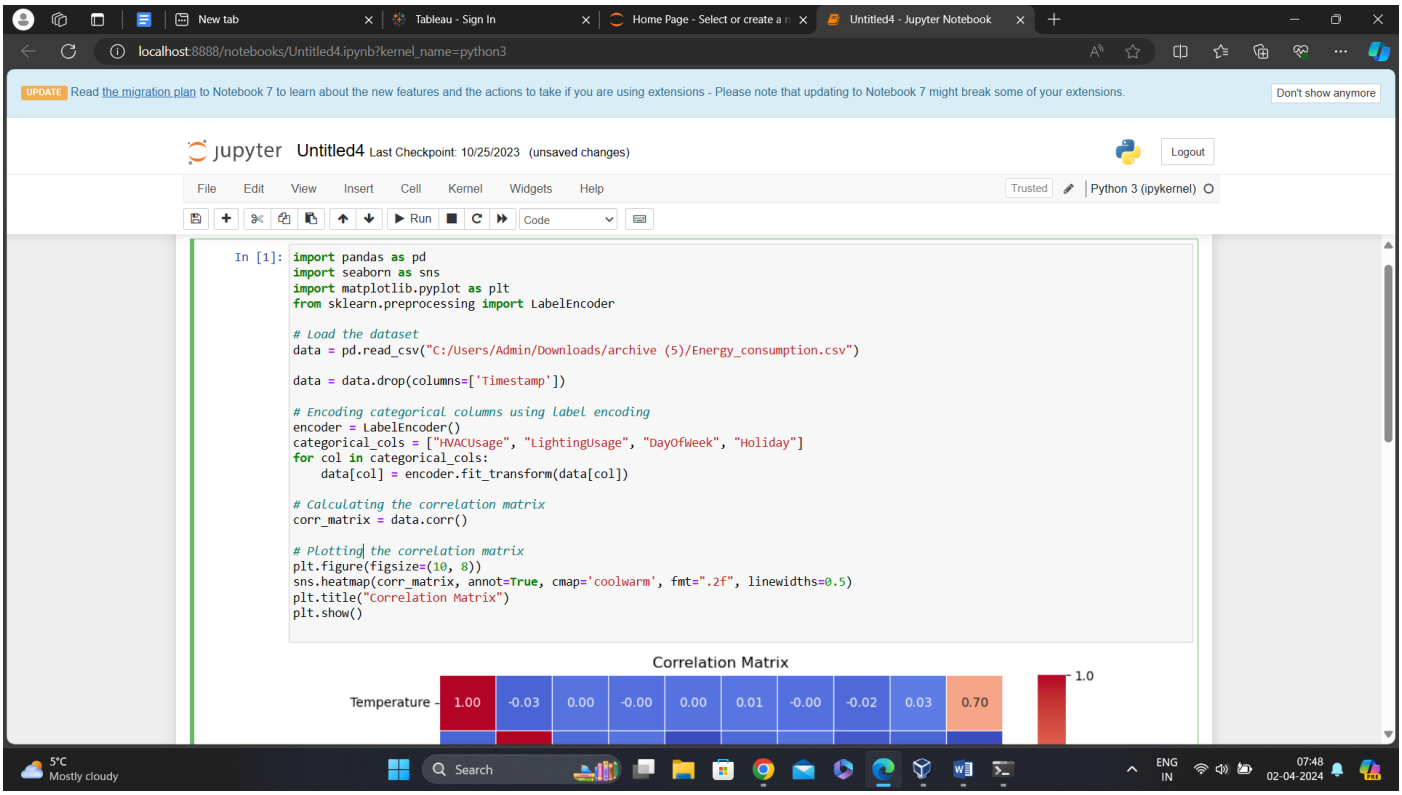
Vaidya, V. (2020, January 3). *Predict Building Energy consumption using Regression Analysis*. Analytics Vidhya. https://medium.com/analytics-vidhya/predict-building-energy-consumption-using-regression-analysis-d63fbbf56e3b

Zhao, H., & Magoulès, F. (2012). A review on the prediction of building energy consumption. *Renewable and Sustainable Energy Reviews*, *16*(6), 3586–3592. https://doi.org/10.1016/j.rser.2012.02.049

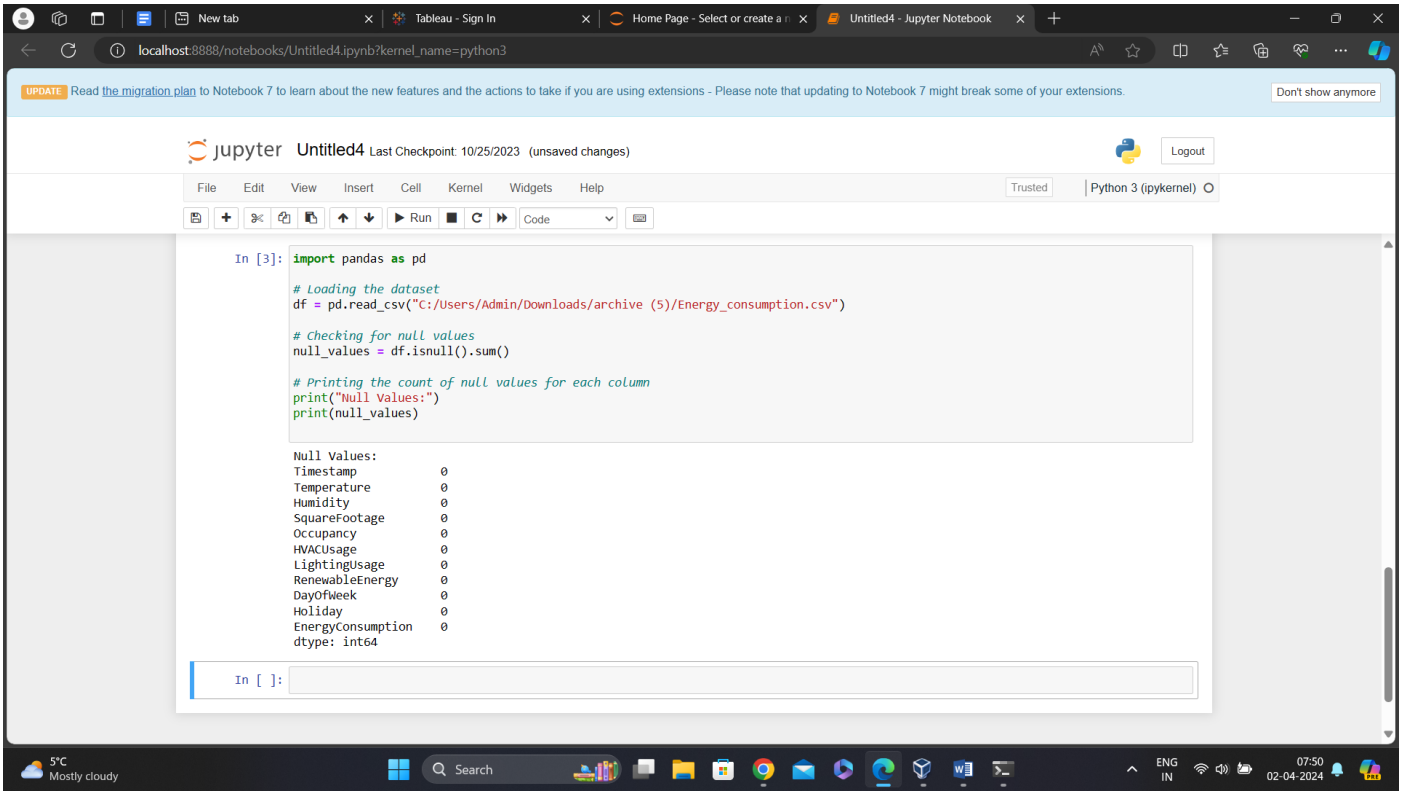
Appendix 1:

Screenshots

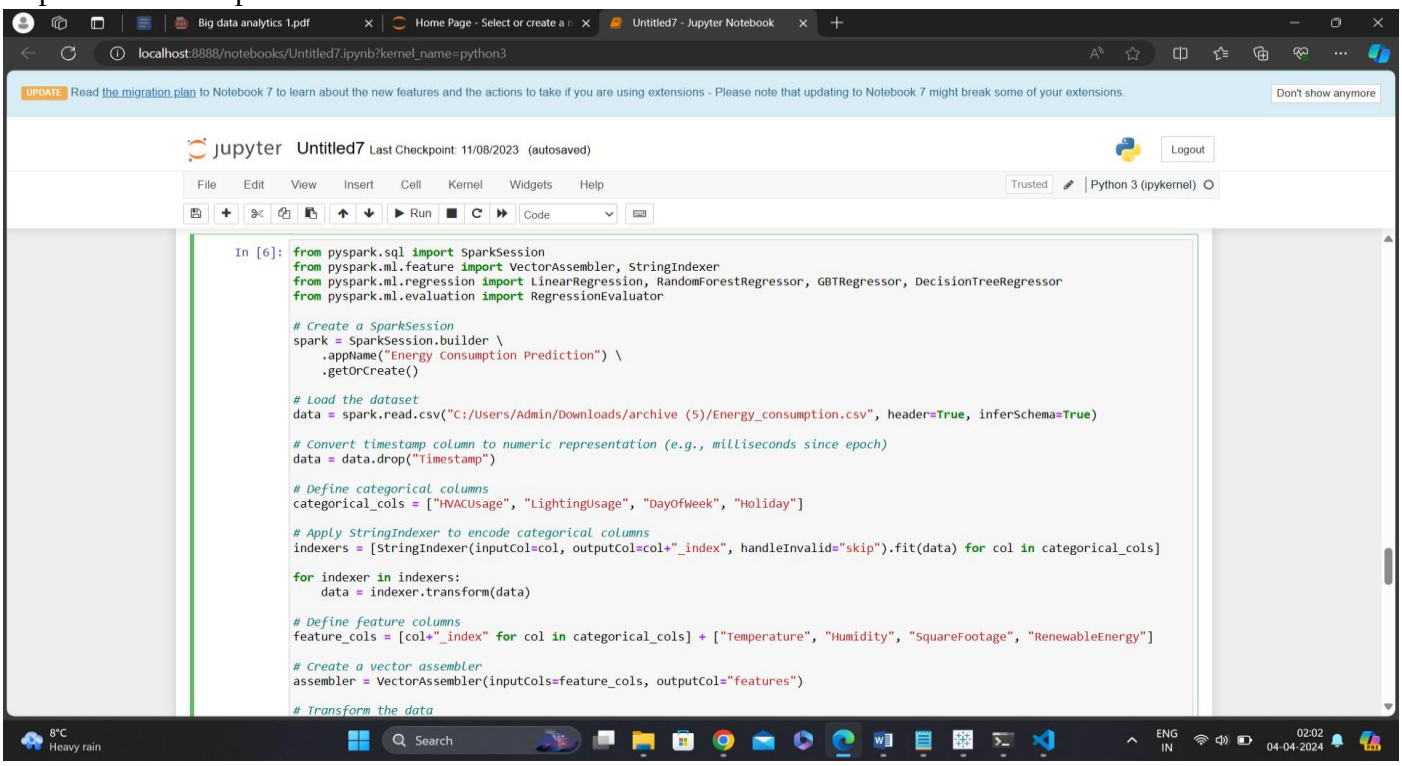
Correlation matrix

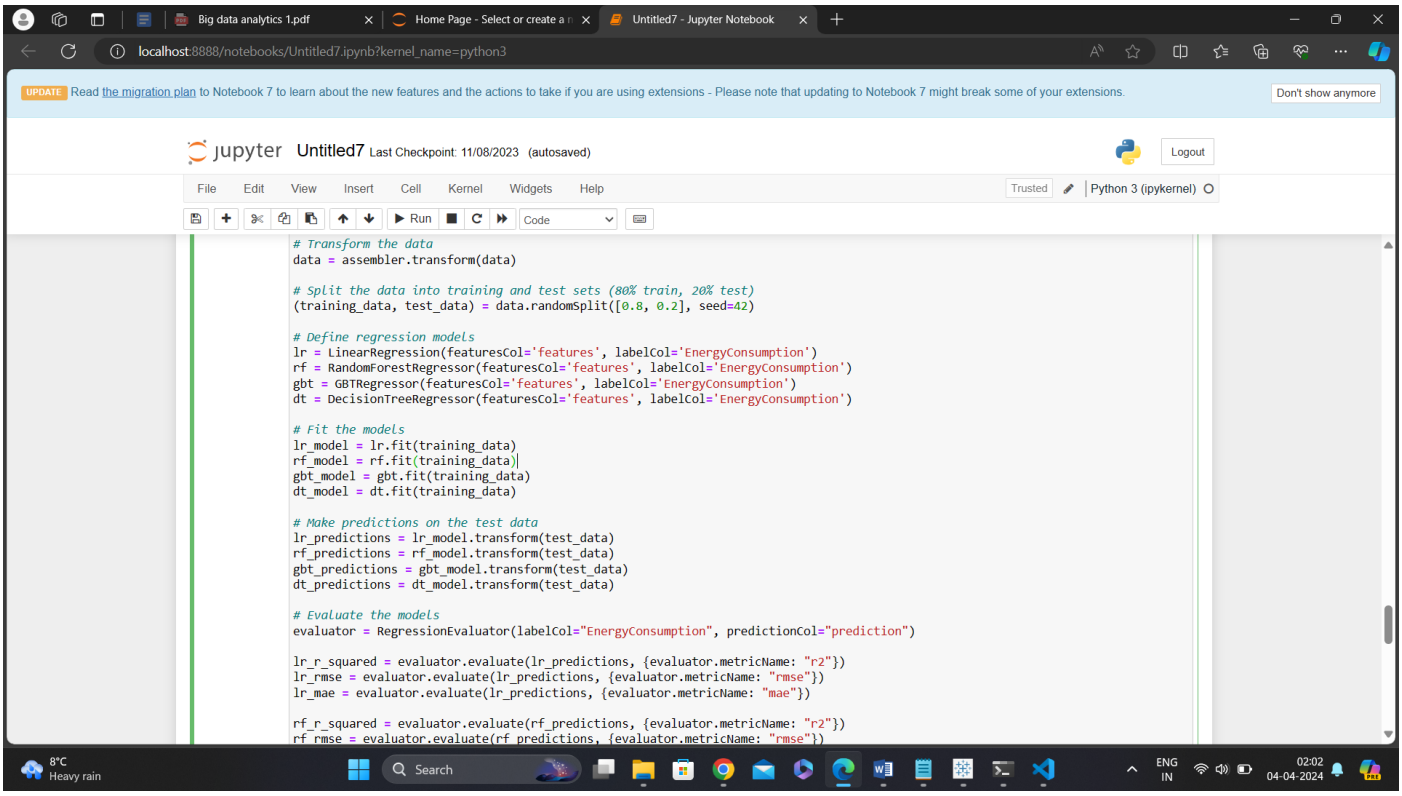


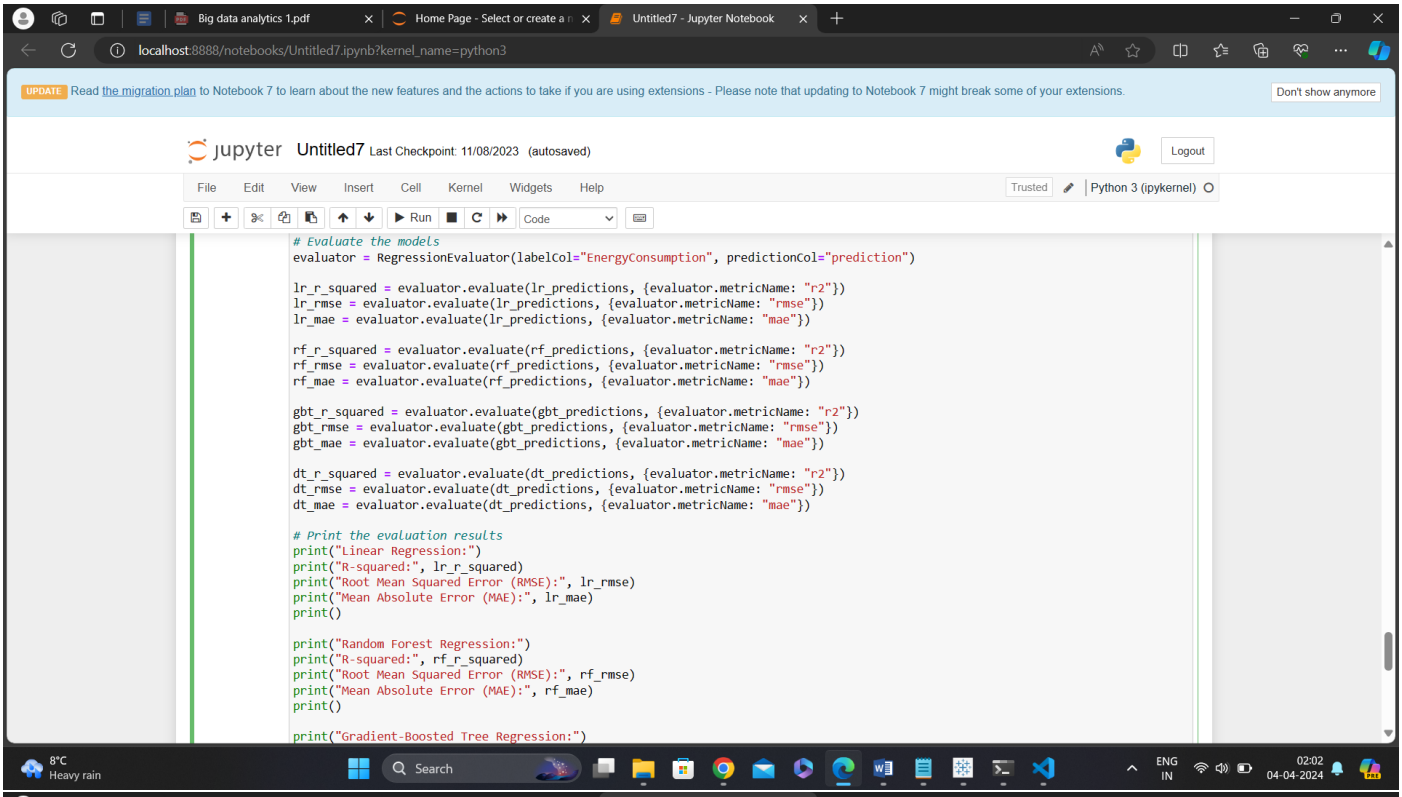
Null value prediction

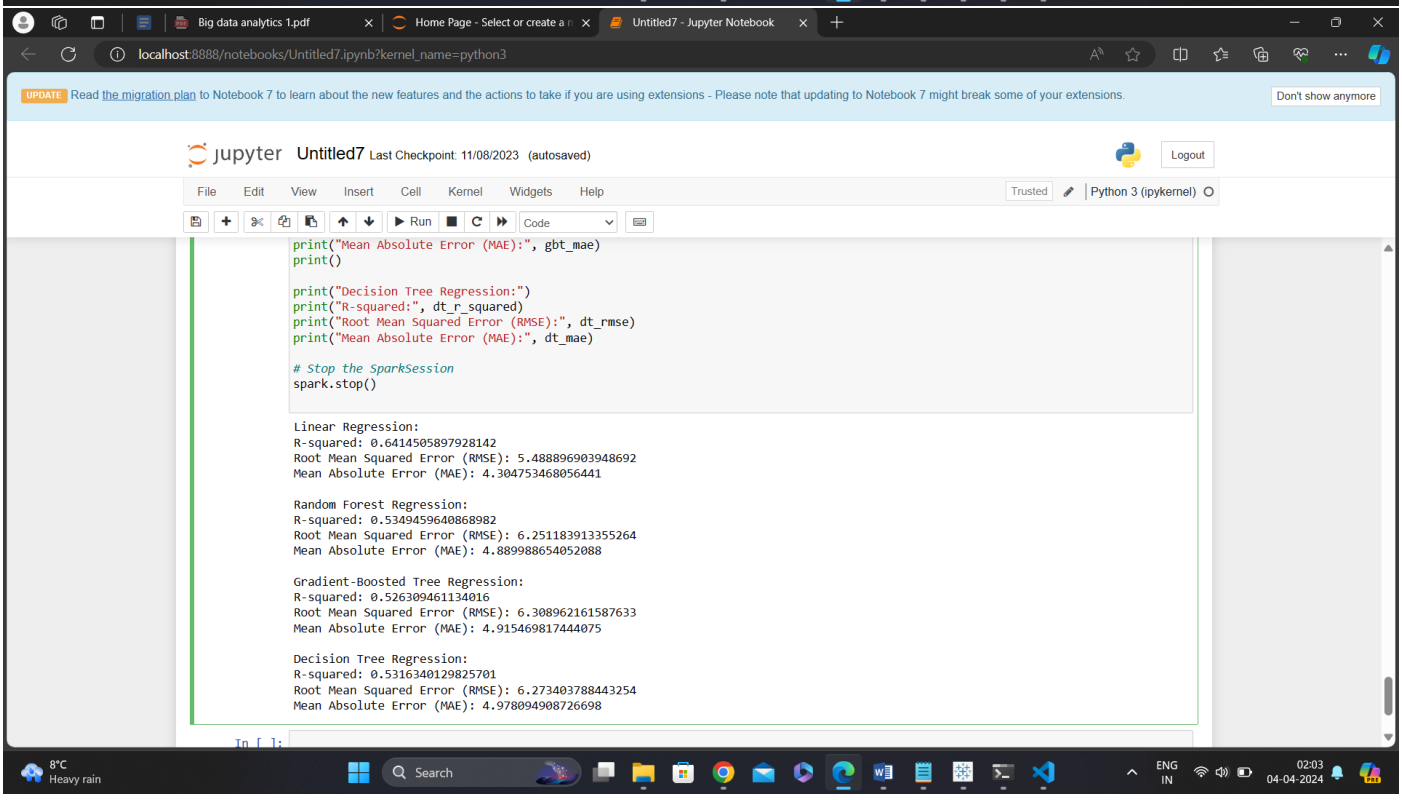


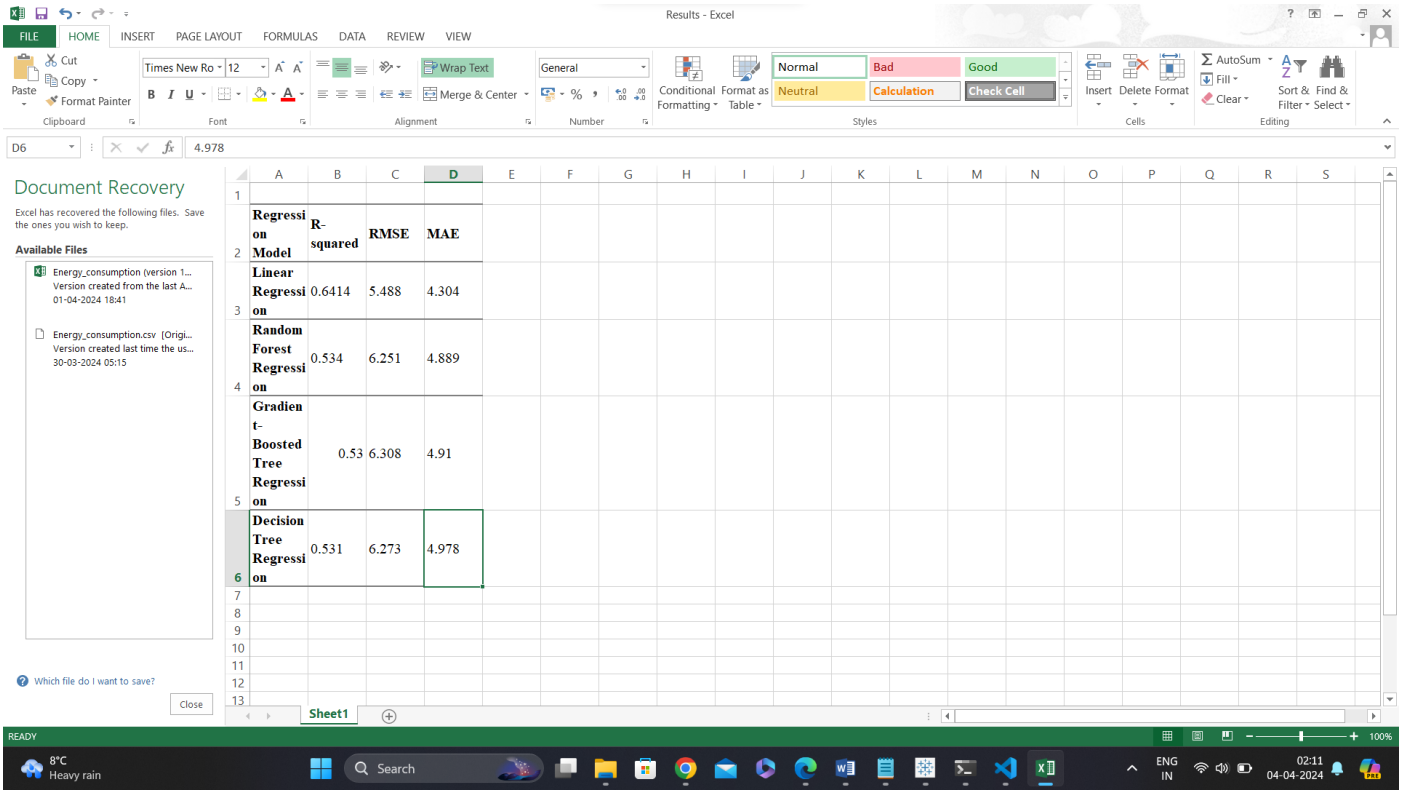
Experimental setup





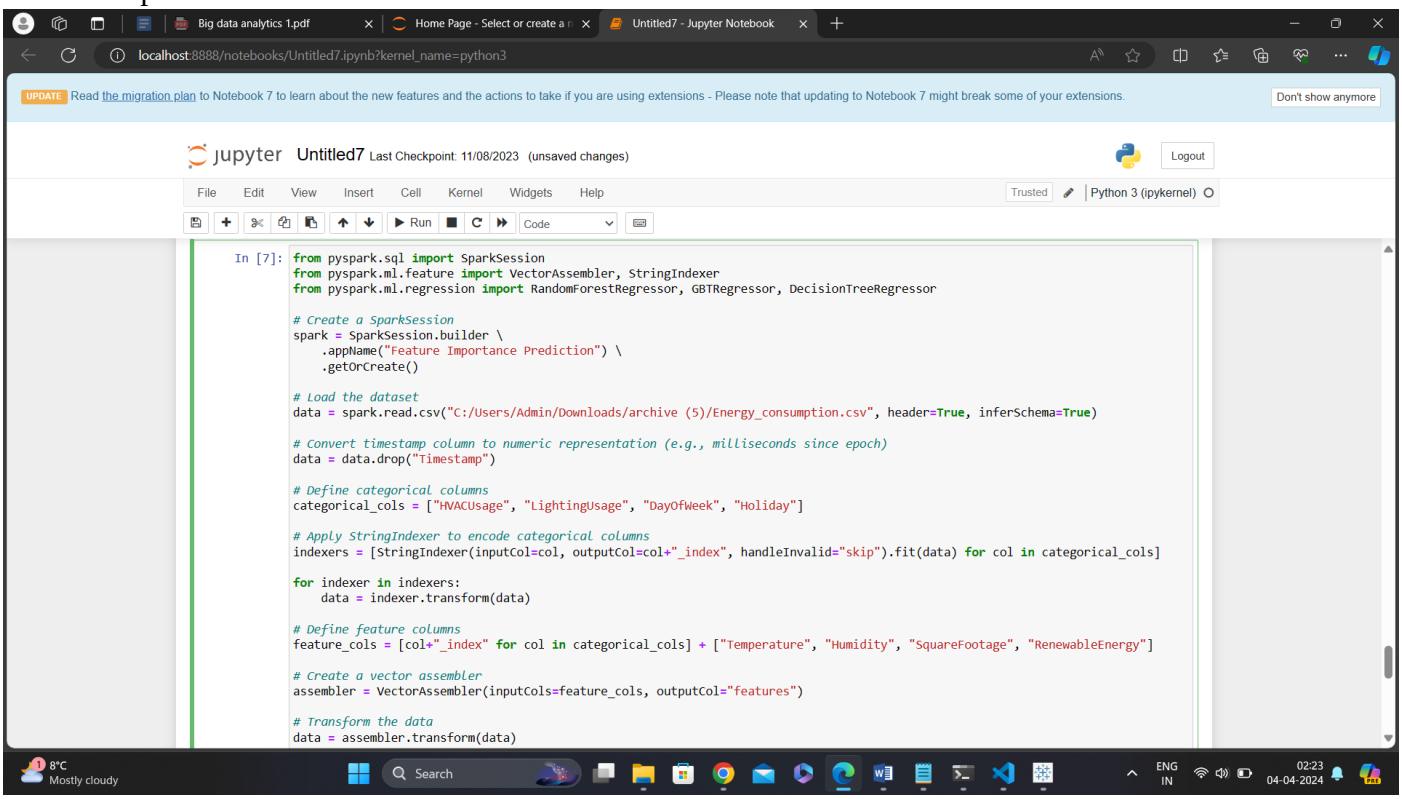

Big data analytics 1.pdf
Home Page - Select or create a n
Untitled7 - Jupyter Notebook
localhost:8888/notebooks/Untitled7.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled7 Last Checkpoint: 11/08/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
# Evaluate the models
evaluator = RegressionEvaluator(labelCol="EnergyConsumption", predictionCol="prediction")
lr_r_squared = evaluator.evaluate(lr_predictions, {evaluator.metricName: "r2"})
lr_rmse = evaluator.evaluate(lr_predictions, {evaluator.metricName: "rmse"})
lr_mae = evaluator.evaluate(lr_predictions, {evaluator.metricName: "mae"})
rf_r_squared = evaluator.evaluate(rf_predictions, {evaluator.metricName: "r2"})
rf_rmse = evaluator.evaluate(rf_predictions, {evaluator.metricName: "rmse"})
rf_mae = evaluator.evaluate(rf_predictions, {evaluator.metricName: "mae"})
gbt_r_squared = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "r2"})
gbt_rmse = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "rmse"})
gbt_mae = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "mae"})
dt_r_squared = evaluator.evaluate(dt_predictions, {evaluator.metricName: "r2"})
dt_rmse = evaluator.evaluate(dt_predictions, {evaluator.metricName: "rmse"})
dt_mae = evaluator.evaluate(dt_predictions, {evaluator.metricName: "mae"})
# Print the evaluation results
print("Linear Regression:")
print("R-squared:", lr_r_squared)
print("Root Mean Squared Error (RMSE):", lr_rmse)
print("Mean Absolute Error (MAE):", lr_mae)
print()
print("Random Forest Regression:")
print("R-squared:", rf_r_squared)
print("Root Mean Squared Error (RMSE):", rf_rmse)
print("Mean Absolute Error (MAE):", rf_mae)
print()
print("Gradient-Boosted Tree Regression:")
8°C Heavy rain
Search
ENG IN
02:02 04-04-2024


Big data analytics 1.pdf
Home Page - Select or create a n
Untitled7 - Jupyter Notebook
localhost:8888/notebooks/Untitled7.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled7 Last Checkpoint: 11/08/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
print("Mean Absolute Error (MAE):", gbt_mae)
print()
print("Decision Tree Regression:")
print("R-squared:", dt_r_squared)
print("Root Mean Squared Error (RMSE):", dt_rmse)
print("Mean Absolute Error (MAE):", dt_mae)
# Stop the SparkSession
spark.stop()
Linear Regression:
R-squared: 0.6414505897928142
Root Mean Squared Error (RMSE): 5.488896903948692
Mean Absolute Error (MAE): 4.304753468056441
Random Forest Regression:
R-squared: 0.5349459640868982
Root Mean Squared Error (RMSE): 6.251183913355264
Mean Absolute Error (MAE): 4.889988654052088
Gradient-Boosted Tree Regression:
R-squared: 0.526309461134016
Root Mean Squared Error (RMSE): 6.308962161587633
Mean Absolute Error (MAE): 4.915469817444075
Decision Tree Regression:
R-squared: 0.5316340129825701
Root Mean Squared Error (RMSE): 6.273403788443254
Mean Absolute Error (MAE): 4.978094908726698
In [ ]:
8°C Heavy rain
Search
ENG IN
02:03 04-04-2024

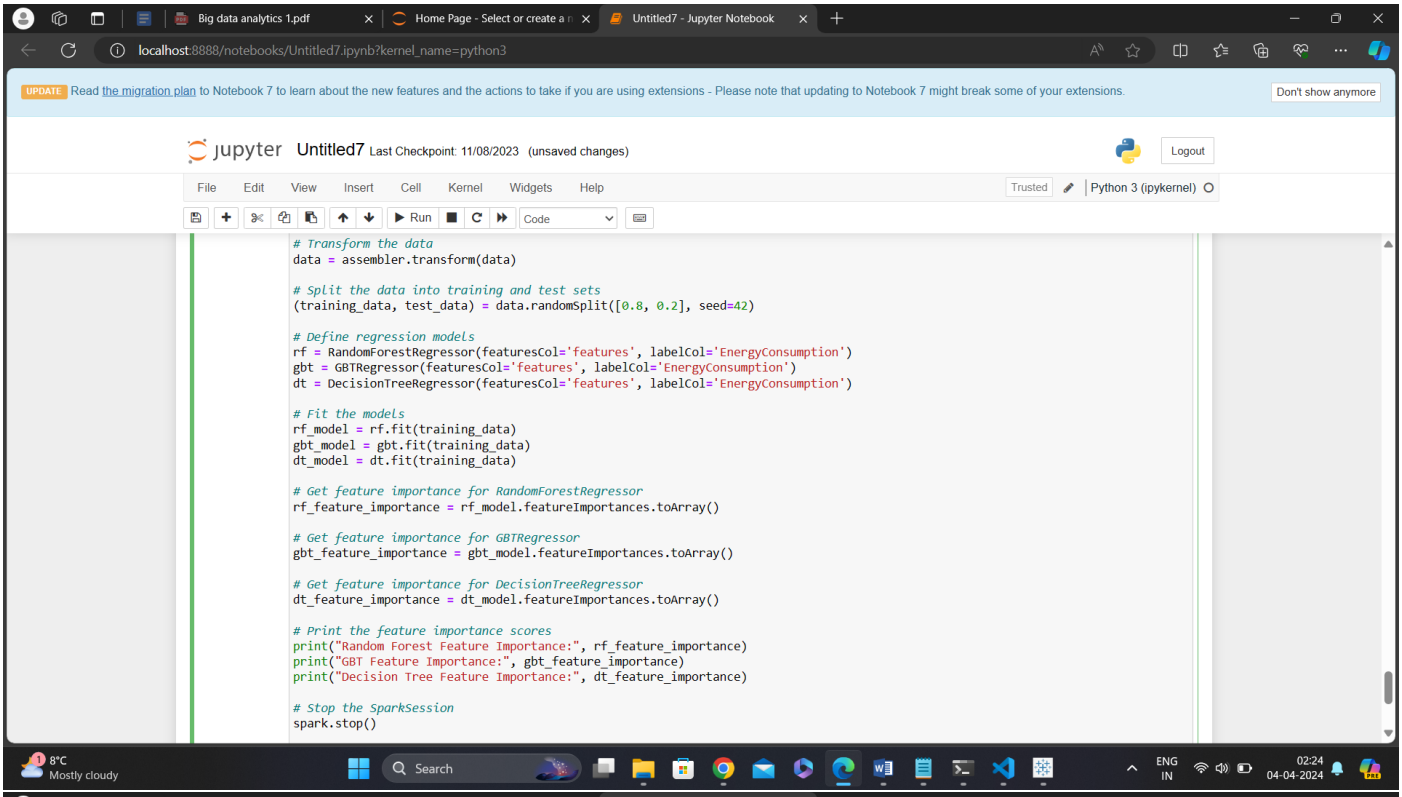


Regression Model	R-squared	RMSE	MAE
Linear Regression	0.6414	5.488	4.304
Random Forest Regression	0.534	6.251	4.889
Gradient-Boosted Tree Regression	0.53	6.308	4.91
Decision Tree Regression	0.531	6.273	4.978

Feature importance:



```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor, DecisionTreeRegressor

# Create a SparkSession
spark = SparkSession.builder \
    .appName("Feature Importance Prediction") \
    .getOrCreate()

# Load the dataset
data = spark.read.csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv", header=True, inferSchema=True)

# Convert timestamp column to numeric representation (e.g., milliseconds since epoch)
data = data.drop("Timestamp")

# Define categorical columns
categorical_cols = ["HVACUsage", "LightingUsage", "DayOfWeek", "Holiday"]

# Apply StringIndexer to encode categorical columns
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index", handleInvalid="skip").fit(data) for col in categorical_cols]

for indexer in indexers:
    data = indexer.transform(data)

# Define feature columns
feature_cols = [col+"_index" for col in categorical_cols] + ["Temperature", "Humidity", "SquareFootage", "RenewableEnergy"]

# Create a vector assembler
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Transform the data
data = assembler.transform(data)
```


Big data analytics 1.pdf
Home Page - Select or create a n
Untitled7 - Jupyter Notebook
localhost:8888/notebooks/Untitled7.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled7 Last Checkpoint: 11/08/2023 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
# Transform the data
data = assembler.transform(data)
# Split the data into training and test sets
(training_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)
# Define regression models
rf = RandomForestRegressor(featuresCol='features', labelCol='EnergyConsumption')
gbt = GBTRegressor(featuresCol='features', labelCol='EnergyConsumption')
dt = DecisionTreeRegressor(featuresCol='features', labelCol='EnergyConsumption')
# Fit the models
rf_model = rf.fit(training_data)
gbt_model = gbt.fit(training_data)
dt_model = dt.fit(training_data)
# Get feature importance for RandomForestRegressor
rf_feature_importance = rf_model.featureImportances.toArray()
# Get feature importance for GBTRegressor
gbt_feature_importance = gbt_model.featureImportances.toArray()
# Get feature importance for DecisionTreeRegressor
dt_feature_importance = dt_model.featureImportances.toArray()
# Print the feature importance scores
print("Random Forest Feature Importance:", rf_feature_importance)
print("GBT Feature Importance:", gbt_feature_importance)
print("Decision Tree Feature Importance:", dt_feature_importance)
# Stop the SparkSession
spark.stop()
8°C Mostly cloudy
Search
ENG IN
02:24 04-04-2024

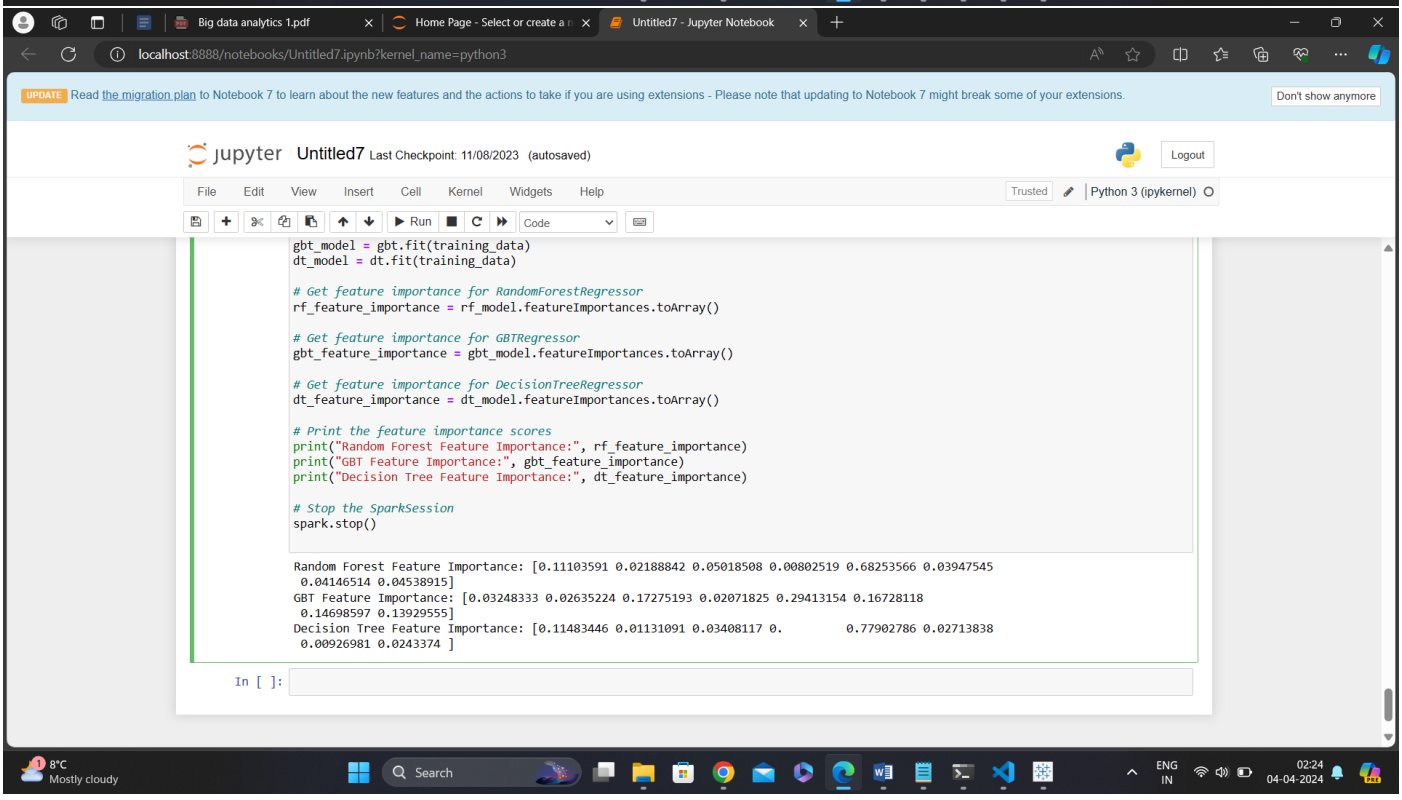

Big data analytics 1.pdf
Home Page - Select or create a n
Untitled7 - Jupyter Notebook
localhost:8888/notebooks/Untitled7.ipynb?kernel_name=python3
UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.
Don't show anymore
jupyter Untitled7 Last Checkpoint: 11/08/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
gbt_model = gbt.fit(training_data)
dt_model = dt.fit(training_data)
# Get feature importance for RandomForestRegressor
rf_feature_importance = rf_model.featureImportances.toArray()
# Get feature importance for GBTRegressor
gbt_feature_importance = gbt_model.featureImportances.toArray()
# Get feature importance for DecisionTreeRegressor
dt_feature_importance = dt_model.featureImportances.toArray()
# Print the feature importance scores
print("Random Forest Feature Importance:", rf_feature_importance)
print("GBT Feature Importance:", gbt_feature_importance)
print("Decision Tree Feature Importance:", dt_feature_importance)
# Stop the SparkSession
spark.stop()
Random Forest Feature Importance: [0.11103591 0.02188842 0.05018508 0.00802519 0.68253566 0.03947545
0.04146514 0.04538915]
GBT Feature Importance: [0.03248333 0.02635224 0.17275193 0.02071825 0.29413154 0.16728118
0.14698597 0.13929555]
Decision Tree Feature Importance: [0.11483446 0.01131091 0.03408117 0. 0.77902786 0.02713838
0.00926981 0.0243374]
In [ ]:
8°C Mostly cloudy
Search
ENG IN
02:24 04-04-2024

Tableau:

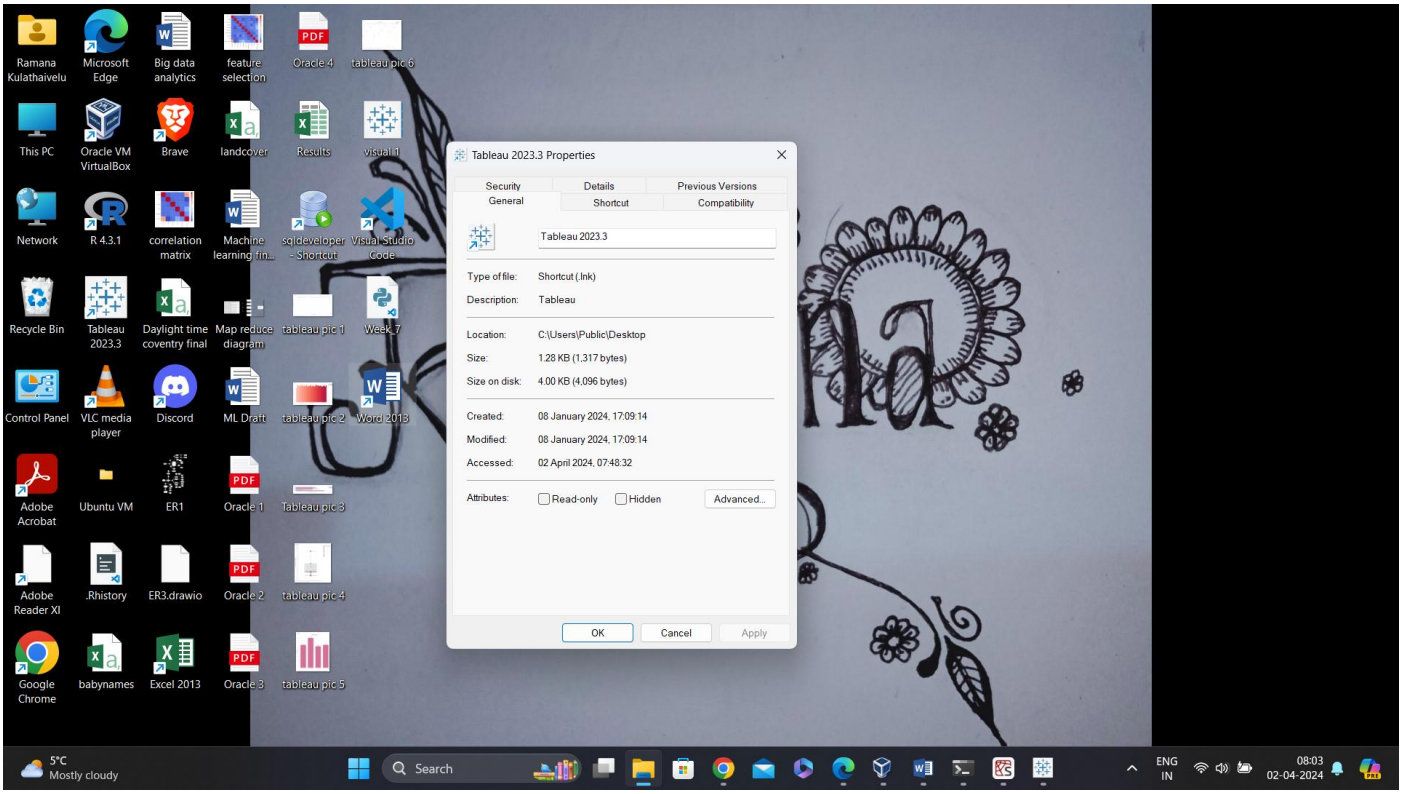


Tableau 2023.3 Properties
Security
Details
Previous Versions
General
Shortcut
Compatibility
Tableau 2023.3
Type of file: Shortcut (.lnk)
Description: Tableau
Location: C:\Users\Public\Desktop
Size: 1.28 KB (1,317 bytes)
Size on disk: 4.00 KB (4,096 bytes)
Created: 08 January 2024, 17:09:14
Modified: 08 January 2024, 17:09:14
Accessed: 02 April 2024, 07:48:32
Attributes: Read-only Hidden Advanced...
OK
Cancel
Apply
Search
ENG IN
08:03 02-04-2024

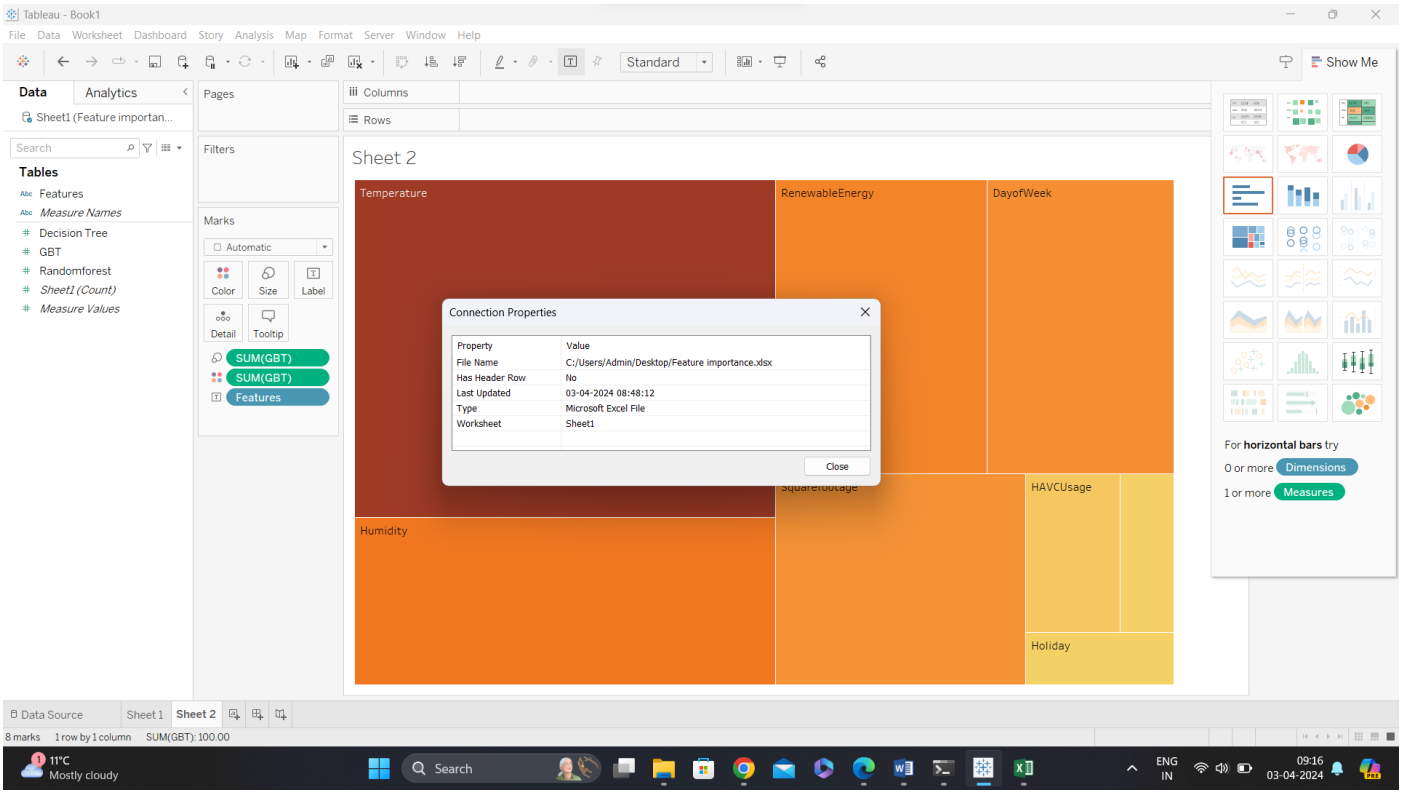


Tableau - Book1
File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Standard
Show Me
Data
Analytics
Sheet1 (Feature importan...
Tables
Features
Measure Names
Decision Tree
GBT
Randomforest
Sheet1 (Count)
Measure Values
Pages
Filters
Marks
Automatic
Color
Size
Label
Detail
Tooltip
SUM(GBT)
SUM(GBT)
Features
Columns
Rows
Sheet 2
Temperature
RenewableEnergy
DayofWeek
HAVCUsage
Humidity
Holiday
Connection Properties
Property
Value
File Name
C:/Users/Admin/Desktop/Feature importance.xlsx
Has Header Row
No
Last Updated
03-04-2024 08:48:12
Type
Microsoft Excel File
Worksheet
Sheet1
Close
For horizontal bars try
0 or more Dimensions
1 or more Measures
Data Source
Sheet 1
Sheet 2
8 marks
1 row by 1 column
SUM(GBT): 100.00
Search
ENG IN
09:16 03-04-2024

Appendix 2:

Correlation matrix

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv")

data = data.drop(columns=['Timestamp'])

# Encoding categorical columns using label encoding
encoder = LabelEncoder()
categorical_cols = ["HVACUsage", "LightingUsage", "DayOfWeek", "Holiday"]
for col in categorical_cols:
    data[col] = encoder.fit_transform(data[col])

# Calculating the correlation matrix
corr_matrix = data.corr()

# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

Counting null values

```
import pandas as pd

# Loading the dataset
df = pd.read_csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv")

# Checking for null values
null_values = df.isnull().sum()

# Printing the count of null values for each column
print("Null Values:")
print(null_values)
```

Experimental setup

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import LinearRegression, RandomForestRegressor,
GBTRegressor, DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator

# Create a SparkSession
spark = SparkSession.builder \
    .appName("Energy Consumption Prediction") \
    .getOrCreate()

# Load the dataset
data = spark.read.csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv",
header=True, inferSchema=True)

# Convert timestamp column to numeric representation (e.g., milliseconds since epoch)
data = data.drop("Timestamp")

# Define categorical columns
categorical_cols = ["HVACUsage", "LightingUsage", "DayOfWeek", "Holiday"]

# Apply StringIndexer to encode categorical columns
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index",
handleInvalid="skip").fit(data) for col in categorical_cols]

for indexer in indexers:
    data = indexer.transform(data)

# Define feature columns
feature_cols = [col+"_index" for col in categorical_cols] + ["Temperature", "Humidity",
"SquareFootage", "RenewableEnergy"]

# Create a vector assembler
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Transform the data
data = assembler.transform(data)

# Split the data into training and test sets (80% train, 20% test)
(training_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)

# Define regression models
lr = LinearRegression(featuresCol='features', labelCol='EnergyConsumption')
rf = RandomForestRegressor(featuresCol='features', labelCol='EnergyConsumption')
gbt = GBTRegressor(featuresCol='features', labelCol='EnergyConsumption')
dt = DecisionTreeRegressor(featuresCol='features', labelCol='EnergyConsumption')

# Fit the models
lr_model = lr.fit(training_data)
rf_model = rf.fit(training_data)
gbt_model = gbt.fit(training_data)
dt_model = dt.fit(training_data)

# Make predictions on the test data
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
dt_predictions = dt_model.transform(test_data)
```

```
# Evaluate the models
evaluator = RegressionEvaluator(labelCol="EnergyConsumption",
predictionCol="prediction")

lr_r_squared = evaluator.evaluate(lr_predictions, {evaluator.metricName: "r2"})
lr_rmse = evaluator.evaluate(lr_predictions, {evaluator.metricName: "rmse"})
lr_mae = evaluator.evaluate(lr_predictions, {evaluator.metricName: "mae"})

rf_r_squared = evaluator.evaluate(rf_predictions, {evaluator.metricName: "r2"})
rf_rmse = evaluator.evaluate(rf_predictions, {evaluator.metricName: "rmse"})
rf_mae = evaluator.evaluate(rf_predictions, {evaluator.metricName: "mae"})

gbt_r_squared = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "r2"})
gbt_rmse = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "rmse"})
gbt_mae = evaluator.evaluate(gbt_predictions, {evaluator.metricName: "mae"})

dt_r_squared = evaluator.evaluate(dt_predictions, {evaluator.metricName: "r2"})
dt_rmse = evaluator.evaluate(dt_predictions, {evaluator.metricName: "rmse"})
dt_mae = evaluator.evaluate(dt_predictions, {evaluator.metricName: "mae"})

# Print the evaluation results
print("Linear Regression:")
print("R-squared:", lr_r_squared)
print("Root Mean Squared Error (RMSE):", lr_rmse)
print("Mean Absolute Error (MAE):", lr_mae)
print()

print("Random Forest Regression:")
print("R-squared:", rf_r_squared)
print("Root Mean Squared Error (RMSE):", rf_rmse)
print("Mean Absolute Error (MAE):", rf_mae)
print()

print("Gradient-Boosted Tree Regression:")
print("R-squared:", gbt_r_squared)
print("Root Mean Squared Error (RMSE):", gbt_rmse)
print("Mean Absolute Error (MAE):", gbt_mae)
print()

print("Decision Tree Regression:")
print("R-squared:", dt_r_squared)
print("Root Mean Squared Error (RMSE):", dt_rmse)
print("Mean Absolute Error (MAE):", dt_mae)

# Stop the SparkSession
spark.stop()
```

Feature importance

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor,
DecisionTreeRegressor

# Create a SparkSession
```

```
spark = SparkSession.builder \
    .appName("Feature Importance Prediction") \
    .getOrCreate()

# Load the dataset
data = spark.read.csv("C:/Users/Admin/Downloads/archive (5)/Energy_consumption.csv",
header=True, inferSchema=True)

# Convert timestamp column to numeric representation (e.g., milliseconds since epoch)
data = data.drop("Timestamp")

# Define categorical columns
categorical_cols = ["HVACUsage", "LightingUsage", "DayOfWeek", "Holiday"]

# Apply StringIndexer to encode categorical columns
indexers = [StringIndexer(inputCol=col, outputCol=col+"_index",
handleInvalid="skip").fit(data) for col in categorical_cols]

for indexer in indexers:
    data = indexer.transform(data)

# Define feature columns
feature_cols = [col+"_index" for col in categorical_cols] + ["Temperature", "Humidity",
"SquareFootage", "RenewableEnergy"]

# Create a vector assembler
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Transform the data
data = assembler.transform(data)

# Split the data into training and test sets
(training_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)

# Define regression models
rf = RandomForestRegressor(featuresCol='features', labelCol='EnergyConsumption')
gbt = GBTRegressor(featuresCol='features', labelCol='EnergyConsumption')
dt = DecisionTreeRegressor(featuresCol='features', labelCol='EnergyConsumption')

# Fit the models
rf_model = rf.fit(training_data)
gbt_model = gbt.fit(training_data)
dt_model = dt.fit(training_data)

# Get feature importance for RandomForestRegressor
rf_feature_importance = rf_model.featureImportances.toArray()

# Get feature importance for GBTRegressor
gbt_feature_importance = gbt_model.featureImportances.toArray()

# Get feature importance for DecisionTreeRegressor
dt_feature_importance = dt_model.featureImportances.toArray()

# Print the feature importance scores
print("Random Forest Feature Importance:", rf_feature_importance)
print("GBT Feature Importance:", gbt_feature_importance)
print("Decision Tree Feature Importance:", dt_feature_importance)

# Stop the SparkSession
spark.stop()
```