ITS66604_0369834_INDVASGNMT

by KAUSHIK RAJ JOSHI .

Submission date: 17-Jul-2025 06:28PM (UTC+0800)
Submission ID: 2716307241
File name: 240852_KAUSHIK_RAJ_JOSHI_._ITS66604_0369834_INDVASGNMT_760881_1351766595.pdf (1.15M)
Word count: 2668
Character count: 15520

TAYLOR'S PROGRAMMES

MAY 2025 SEMESTER

ITS66604 - Machine Learning and Parallel Computing

Instruction to Candidates:

1. Name your answer file as ITS69304_XXXXXX_INDVASGNMT.pdf where XXXXXX is your STUDENT NUMBER. Then, submit to the MyTIMeS portal via the link "**INDIVIDUAL ASSIGNMENT submission**" on the module page. (Do not submit the question paper).

Assignment No./Title	**Assignment (Individual Assignment)** **20% Weightage**
Course Tutor/Lecturer	Mr. Anmol Adhikari, Mr. Bidhan Chandra Bhattarai
Submission Date	21st July 2025
Student Name, ID and Signature 1. Kaushik Raj Joshi – 0369834 	

1

Declaration *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Marks/Grade:	**Evaluated by:**
Evaluator's Comments:	

* Please include this cover page for your project submission

Table of Contents

Table of Contents 3

Problem Context and Dataset Justification 4

System Design for Scalable Machine Learning Solution 8

Implementation Strategy and Experimentation 17

Ethical and Professional Responsibility 23

System Architecture and Deployment Planning 25

Professional Communication 26

References 27

Problem Context and Dataset Justification

Wildfires are one of the most disastrous and prevalent natural disasters in the world, which leads to the loss of life, economic damage, negative and long-term climate impact and destruction of ecosystems. With the rising temperatures and droughts, the frequency and the severity of wildfires has increased a lot in recent years.

Wildfire prediction is and resource planning are very crucial for:

i) Preventing loss of human and animal life
ii) To protect areas where people live
iii) To manage the evacuation procedures
iv) Utilize firefighting resources and equipment efficiently
v) Reducing environmental damage and pollution.

Traditional methods for wildfire protection use historical patterns, but in the modern age we have high availability of large scale data and also various machine learning algorithms. With ML we can develop predictive models that can forecast wildfire size, severity, frequency and probability. In addition, handling large datasets effectively and efficiently requires parallel computing technology as well, especially in real time data.

Dataset Selection and Justification:

For this project, we will be using the '2000-2021 TURKEY FIRE POINTS / SINGLE CSV / NASA' dataset from Kaggle to predict Fire Radioactive Power (FRP) of wildfires in Turkey. This dataset contains detailed information, including:

- Latitude, Longitude – Spatial location
- Brightness – Thermal intensity from satellite sensors
- Scan and Track – Satellite sensor resolution in x and y axes
- Confidence – Confidence on detection which ranges from 0-100%
- FRP – Variable indicating fire intensity
- Bright_t31 – Brightness temperature at 31μm
- Acquisition Date & Time – When fire got detected

Here Bright_t31 refers to Brightness temperature at 31μm (31 micrometers) since it is a part of the infrared spectrum which is sensitive to surface heat.

The reasons for choosing this dataset are:

i) Real Disaster Data: This dataset is based on satellite observation from Turkey wildfires, making it directly relevant to its environment management
ii) Prediction: Variables like brightness, scan, track, and confidence can be used to predict FRP
iii) Public Safety: Predicting fires will help for strategic firefighting planning
iv) Data Volume: Since it contains over 20 years of data the dataset is large and suitable for parallel processing techniques to speed up computation

The challenges in this dataset include:

i) Data Imbalance: Many fires are small with low FRP, but a few extreme cases dominate the data
ii) Outliers: Extreme values in the dataset should be handled properly
iii) Noise: Because of variation in scan and track values, it may contain noise because of sensor limitations in satellite
iv) Computation Power: If this data is extended with weather, temperature the dataset will take more computational power.

Preprocessing and Data Exploration:

To preprocess the data, following steps were used:

1) Handling missing values: Dropped the rows with missing values

```
# Handle missing values
df = df.dropna()
```

2) Remove outliers: Removed outliers using IQR

1
IQR = Q3-Q1

Q3 = median of 3rd quartile

Q1 = median of 1st quartile

To find the outliers,

Lower bound = Q1 - 1.5 * IQR

Upper bound = Q3 + 1.5 * IQR

If anything is below lower bound or above upper bound, it is an outlier and is removed

```
# Remove outliers using IQR
# Calculate Q1 and Q3 for numerical columns
numerical_cols = df.select_dtypes(include=np.number).columns.tolist()

Q1 = df[numerical_cols].quantile(0.25)
Q3 = df[numerical_cols].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df_filtered = df[~((df[numerical_cols] < lower_bound) | (df[numerical_cols] > upper_bound)).any(axis=1)]

print(f"Original data shape: {df.shape}")
print(f"Filtered data shape: {df_filtered.shape}")

df = df_filtered.copy() # Copy the data with no outliers
```

```
Original data shape: (205918, 15)
Filtered data shape: (133412, 15)
```

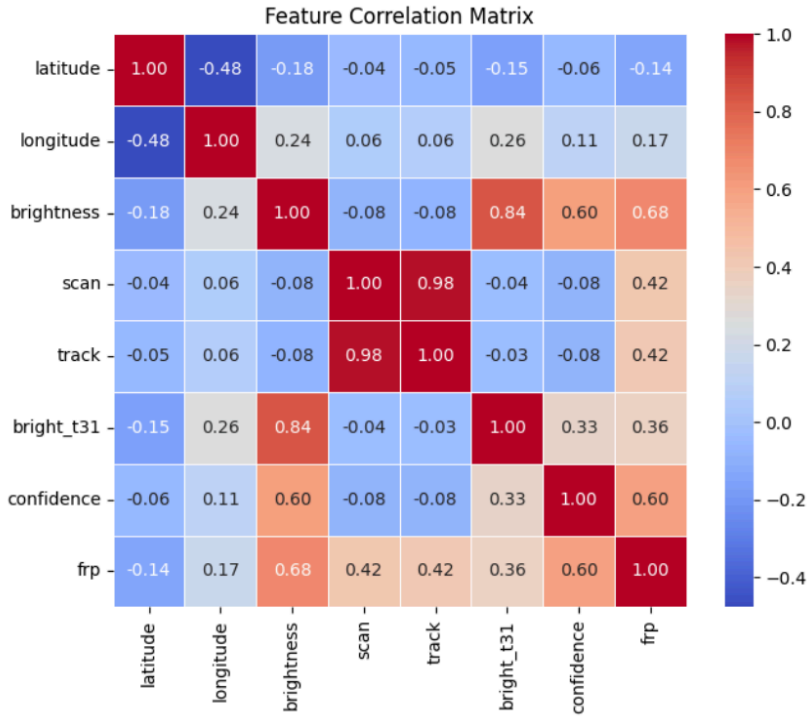
3) Exploratory Analysis: Generated a correlation heatmap to find relationships between columns which may be helpful in finding FRP

```
# Compute correlation matrix
corr = df[['latitude', 'longitude', 'brightness', 'scan', 'track', 'bright_t31', 'confidence', 'frp']].corr()

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True, linewidths=0.5)
plt.title("Feature Correlation Matrix")
plt.show()
```


```
# Compute correlation matrix
corr = df[['latitude', 'longitude', 'brightness', 'scan', 'track', 'bright_t31', 'confidence', 'frp']].corr()

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True, linewidths=0.5)
plt.title("Feature Correlation Matrix")
plt.show()
```



4) Feature Selection: Selecting columns to predict FRP (brightness, scan, track, and confidence)

```
# Features
features = ['brightness', 'scan', 'track','confidence']
target = 'frp'
```

System Design for Scalable Machine Learning Solution

The primary goal of this project is to predict Fire Radiative Power (FRP) of wildfires using features like brightness, scan, track, and confidence.

Machine Learning models:

i) Linear Regression

ii) Random Forest

Evaluation Matrices:

Mean Absolute Error (MAE): MAE is the average of absolute difference between actual and predicted value.

Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Where,

n = number of observations

y_i = true value

ŷ_i = predicted value (Mean Absolute Error(MSE), n.d.)

Root Mean Squared Error (RMSE): Measures the average magnitude of errors.

Formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\|y(i) - \hat{y}(i)\|^2}{N}},$$

Where,

n = number of observations

y_i = true value

$\hat{y}_i$ = predicted value (What is Root Mean Square Error?, n.d.)

R^2: R^2 shows how much the variance in the actual value is explained by the model. It ranges from 0-1.

Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Where,

n = number of observations

y_i = true value

$\hat{y}_i$ = predicted value (Kumar, 2023)

Linear Regression:

Linear Regression is a ML algorithm which learns from a dataset and maps data points with optimal linear functions to be used for future prediction on new datasets. It assumes relationship between the independent variables (features) and dependent variables (target). (Linear Regression in Machine learning, 2025)

Linear Regression will form a baseline for the FRP prediction. Linear regression is simple, fast to train and easy to interpret. The drawback for linear regression is that it is less robust for outliers and complex patterns.

Since this is a multiple linear regression:

Formula = $h(x_1,x_2,...,x_k) = \beta_0 + \beta_1x_1 + \beta_2x_2 + ... + \beta_kx_k$

Where,

$x_1,x_2,...,x_k$ = independent variables

β_0 = Intercept

$\beta_1,\beta_2,\ldots,\beta_k$ = coefficients which represents the influence of each feature on the predicted output.

(Linear Regression in Machine learning, 2025)

In our case,

FRP= β0 + β1 × brightness + β2 × scan + β3× track + β4 ×confidence

Where e = error term

Code:

```
[20] # Select features and target

     features = ['brightness', 'scan', 'track','confidence']
     target = 'frp'

     X = df[features]
     y = df[target]
```

```
[21] # Split data into train and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

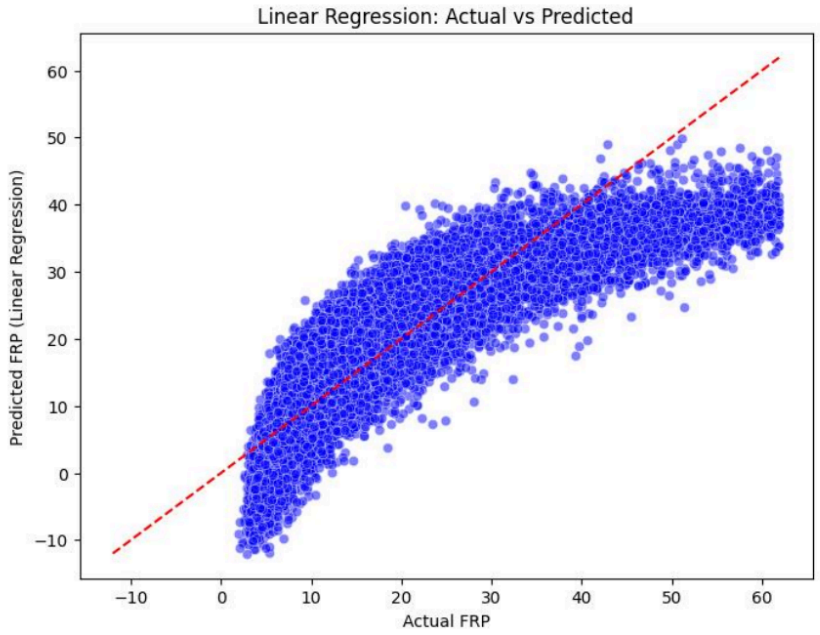
```
# Linear Regression Model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict
y_pred_lr = lr.predict(X_test)

# Evaluate Linear Regression
print("Linear Regression Performance:")
print("MAE:", mean_absolute_error(y_test, y_pred_lr))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_lr)))
print("R2 Score:", r2_score(y_test, y_pred_lr))
```

```
Linear Regression Performance:
MAE: 4.727006291029031
RMSE: 6.1745663718829285
R2 Score: 0.7549565415424608
```

```python
# Plot Linear Regression Results
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred_lr, alpha=0.5, color="blue")
# Plot perfect prediction line
min_val = min(y_test.min(), y_pred_lr.min())
max_val = max(y_test.max(), y_pred_lr.max())
plt.plot([min_val, max_val], [min_val, max_val], '--', color='red', label='Perfect Prediction Line')
plt.xlabel("Actual FRP")
plt.ylabel("Predicted FRP (Linear Regression)")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
```



Interpretation:

MAE: On average, the predictions are off by 4.72 units

RMSE: On average, the predictions deviate from actual value by 6.17 units

R^2: About 75.4% of the variance in actual value is explained by the model. Statistically we can say that it is a good model

Random Forest:

Random Forest is a ML algorithm that uses decision trees to make predictions. It combines the decisions of multiple decision trees for classification or regression. (Random Forest Algorithm in Machine Learning, 2025)

Random Forest is able to handle high dimensional data well and is robust to outliers and noise. The drawback for random forest is that it is more computationally intensive.

Code:

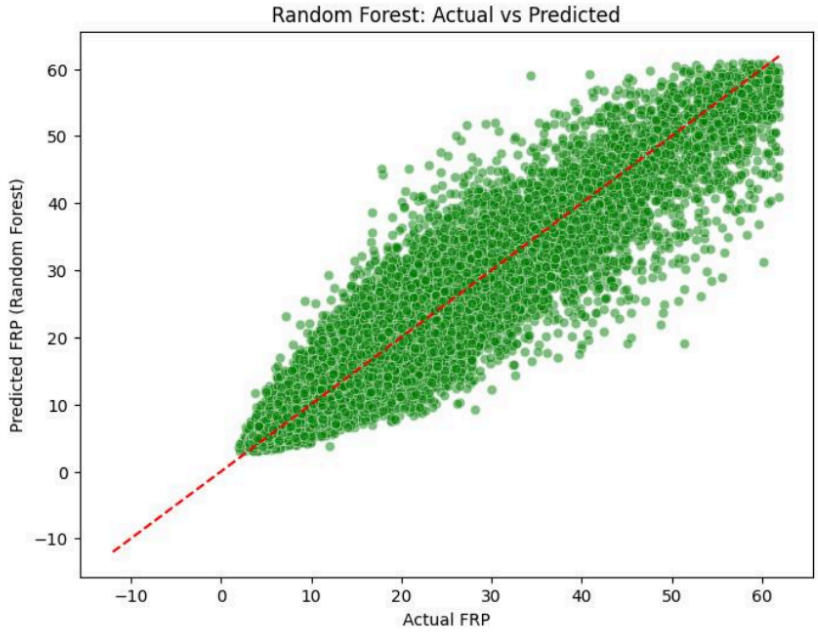
```
# Random Forest Model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)

# Evaluate Random Forest
print("Random Forest Performance:")
print("MAE:", mean_absolute_error(y_test, y_pred_rf))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_rf)))
print("R2 Score:", r2_score(y_test, y_pred_rf))
```

```
Random Forest Performance:
MAE: 2.822545196205912
RMSE: 4.090959301059221
R2 Score: 0.8924326413419457
```

```
# Plot Random Forest Results
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred_rf, alpha=0.5, color="green")
plt.plot([min_val, max_val], [min_val, max_val], '--', color='red', label='Perfect Prediction Line')
plt.xlabel("Actual FRP")
plt.ylabel("Predicted FRP (Random Forest)")
plt.title("Random Forest: Actual vs Predicted")
plt.show()
```



Interpretation:

MAE: On average, the predictions are off by 2.82 units

RMSE: On average, the predictions deviate from actual value by 4.09 units

R^2: About 89.2% of the variance in actual value is explained by the model. Statistically we can say that it is a very good model

Parallel Processing:

Handling large-scale data requires efficient data preprocessing, especially when working with millions of records collected over multiple years.

Advantages of PySpark:

i) Scalability: PySpark can process datasets by distributing its data and computations to multiple nodes within a cluster. It also scales horizontally so we can add more machines as required

ii) Ease of Use: We can use Spark code in Python with PySpark, making it very accessible

iii) Integration in Big Data: PySpark can easily integrate in big data ecosystems like HADOOP, Kafka etc. (DataWithSantosh, 2025)

Here we used PySpark which uses Apache Spark technology to address the challenge.

i) Distributed data handling: PySpark's dataframe API allows CSV files to be processed across multiple cores.

ii) Aggregation: Used spark_df.agg(sum('brightness')) to calculate brightness in parallel

iii) Conversion to Pandas: After performing the distributed computations, results are converted back into pandas.

```
# Create Spark session
spark = SparkSession.builder.appName("TurkeyFireAnalysis").getOrCreate()

spark_df= spark.createDataFrame(df)
spark_df.show()
```

```
+--------+---------+----------+----+-----+----------+--------+---------+----------+----------+-------+----------+----+-------+----+
|latitude|longitude|brightness|scan|track|  acq_date|acq_time|satellite|instrument|confidence|version|bright_t31| frp|daynight|type|
+--------+---------+----------+----+-----+----------+--------+---------+----------+----------+-------+----------+----+-------+----+
| 36.8527|  35.3594|     306.0| 2.1|  1.4|2000-11-01|     815|    Terra|     MODIS|        60|   6.03|     294.4|13.2|       D| 0.0|
| 40.5441|  35.1786|     307.8| 1.1|  1.0|2000-11-02|     857|    Terra|     MODIS|        66|   6.03|     296.0| 5.8|       D| 0.0|
| 40.1473|  35.0444|     305.3| 1.1|  1.0|2000-11-02|     857|    Terra|     MODIS|        46|   6.03|     293.5| 4.3|       D| 0.0|
| 40.5421|  35.1909|     310.0| 1.1|  1.0|2000-11-02|     857|    Terra|     MODIS|        69|   6.03|     296.1| 7.3|       D| 0.0|
| 39.1096|  33.9391|     324.5| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        82|   6.03|     294.4|20.2|       D| 0.0|
| 39.1981|  32.4182|     308.9| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        64|   6.03|     294.6| 6.1|       D| 0.0|
| 39.8424|  32.4918|     306.9| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        64|   6.03|     289.9| 5.8|       D| 0.0|
| 39.1002|  33.9423|     328.3| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        85|   6.03|     294.3|25.9|       D| 0.0|
| 39.3864|   32.391|     306.2| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        49|   6.03|     294.8| 4.1|       D| 0.0|
| 39.3395|  32.2664|     326.7| 1.0|  1.0|2000-11-02|     858|    Terra|     MODIS|        84|   6.03|     293.9|23.2|       D| 0.0|
```

```
# Perform parallel aggregation (sum) on the 'brightness' column
total_brightness = spark_df.agg(sum('brightness')).collect()[0][0]

print(f"Total Brightness: {total_brightness}")
```

```
Total Brightness: 43095857.200000085
```

```
[52] # Extract the year from 'acq_date' and create a new 'Year' column
     spark_df_with_year = spark_df.withColumn("Year", year("acq_date"))

     # Group by 'Year' and calculate the sum of 'brightness' for each year
     final_result = spark_df_with_year.groupBy("Year").agg(sum("brightness").alias("total_brightness_by_year"))

     # Show the result
     final_result.orderBy("Year").show()
```

```
+----+------------------------+
|Year|total_brightness_by_year|
+----+------------------------+
|2000|      144385.09999999995|
|2001|       1076852.200000003|
|2002|       1468170.900000001|
|2003|      1602603.1999999997|
|2004|      1815037.6000000031|
|2005|       1791790.000000001|
|2006|      1912221.1000000015|
|2007|       1770349.800000001|
|2008|       1240878.200000004|
|2009|      3881693.4999999977|
|2010|       2773047.800000015|
|2011|       3911255.799999999|
|2012|      1499348.8000000012|
|2013|        2445505.60000001|
|2014|      1418871.4999999988|
|2015|             3668936.3|
|2016|      2947289.4000000004|
|2017|      1945874.3000000073|
|2018|      1758386.6999999958|
|2019|      1980234.5999999929|
+----+------------------------+
only showing top 20 rows
```

Here PySpark splits the dataset into different parts to distribute the competition across the cores. This results in the computation being faster and more efficient in larger datasets.

Tools Used:

Scikit-learn: Model Training for linear regression and random forest

Joblib: Enables parallel tree construction in Random Forest

PySpark: Enables using Spark in Python

System Design:

- Data Collection: Data was collected from Kaggle and loaded
- Data Preprocessing: Applied IQR in numerical columns to remove the outliers and in parallel computing, we used sum to group operations

- Model Training: Model was trained on Linear Regression and Random Forest Regression
- Model Evaluation: Evaluated the model with R^2, RMSE, MAE and visualized actual vs predicted
- Model Deployment

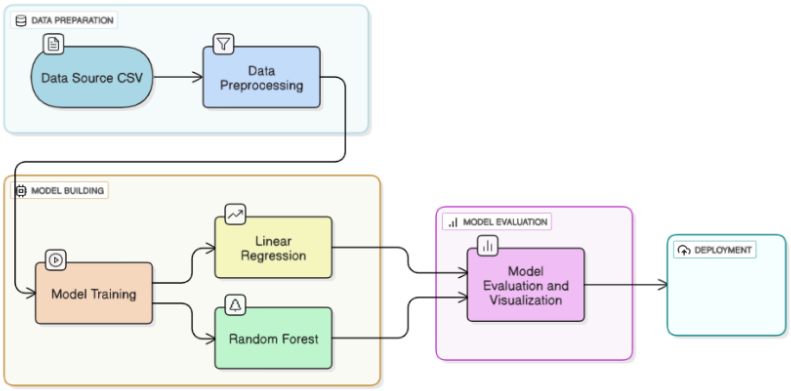


Fig: System Design

Implementation Strategy and Experimentation

Here we used 2 models for FRP Prediction; linear regression, random forest.

Data was split in:

Train: 80%

Test: 20%

Model Training (Parallelly):

Handling data from 2000-2021 will involve thousands of data collected over time. Without parallelization processing datasets will take a lot of time.

How Parallel Computing is utilized:

i) We utilize PySpark for parallel aggregation in sum('brightness')) using spark_df.agg() and other summary statistics

- PySpark distributes computations across cores and clusters
- Scales well with massive datasets

```
total_brightness = spark_df.agg(sum('brightness')).collect()[0][0]

print(f"Total Brightness: {total_brightness}")
```

```
Total Brightness: 43095857.200000085
```

```
# Extract the year from 'acq_date' and create a new 'Year' column
spark_df_with_year = spark_df.withColumn("Year", year("acq_date"))

# Group by 'Year' and calculate the sum of 'brightness' for each year
final_result = spark_df_with_year.groupBy("Year").agg(sum("brightness").alias("total_brightness_by_year"))

# Show the result
final_result.orderBy("Year").show()
```

```
+----+------------------------+
|Year|total_brightness_by_year|
+----+------------------------+
|2000|      144385.09999999995|
|2001|       1076852.200000003|
|2002|       1468170.900000001|
|2003|      1602603.1999999997|
|2004|      1815037.6000000031|
|2005|       1791790.000000001|
|2006|      1912221.1000000015|
|2007|       1770349.800000001|
|2008|       1240878.200000004|
|2009|      3881693.4999999977|
|2010|       2773047.800000015|
|2011|       3911255.799999999|
|2012|      1499348.8000000012|
|2013|        2445505.60000001|
|2014|      1418871.4999999988|
```

ii) Training the models in parallel

```
# Modify Linear Regression Training and Prediction with joblib
print("Linear Regression Performance (Parallel):")
lr_parallel = LinearRegression()
lr_parallel.fit(X_train, y_train)

# Parallel prediction
n_cores = -1  # Use all available CPU cores
y_pred_lr_parallel = Parallel(n_jobs=n_cores)(delayed(lr_parallel.predict)(X_test.iloc[i:i+1])
for i in range(len(X_test)))
y_pred_lr_parallel = np.concatenate(y_pred_lr_parallel)


# Evaluate Linear Regression
print("MAE:", mean_absolute_error(y_test, y_pred_lr_parallel))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_lr_parallel)))
print("R2 Score:", r2_score(y_test, y_pred_lr_parallel))
```

Parallel training is not supported by scikit-learn's LinearRegression. We can parallelize the prediction but not the fit. As parallel fit for Linear Regression is not supported, we will still use sequential to fit the time.

Here, n_cores = -1 is used for utilizing all of the cores

```
print("\nRandom Forest Performance (Parallel):")
rf_parallel = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=n_cores)
rf_parallel.fit(X_train, y_train)

# Parallel prediction (RandomForestRegressor's predict also supports n_jobs)
y_pred_rf_parallel = rf_parallel.predict(X_test)

# Evaluate Random Forest
print("MAE:", mean_absolute_error(y_test, y_pred_rf_parallel))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_rf_parallel)))
print("R2 Score:", r2_score(y_test, y_pred_rf_parallel))
```

In Random Forest we modify the training and prediction with joblib since Random Forest Regressor supports n_jobs for parallel training.

iii) Evaluation:

```
Linear Regression Performance (Parallel):
MAE: 4.7270062910290305
RMSE: 6.1745663718829285
R2 Score: 0.7549565415424608
```

Linear Regression:

MAE: On average, the predictions are off by 4.72 units

RMSE: On average, the predictions deviate from actual value by 6.17 units

R^2: About 75.4% of the variance in actual value is explained by the model. Statistically we can say that it is a good model

```
Random Forest Performance (Parallel):
MAE: 2.822545196205912
RMSE: 4.090959301059221
R2 Score: 0.8924326413419457
```

Random Forest:

MAE: On average, the predictions are off by 2.82 units

RMSE: On average, the predictions deviate from actual value by 4.09 units

R^2: About 89.2% of the variance in actual value is explained by the model. Statistically we can say that it is a very good model

iv) Training model in sequential:

```
# Calculate and print accuracy metrics for sequential Linear Regression
print("Linear Regression Performance (Sequential):")
mae_lr_sequential = mean_absolute_error(y_test, y_pred_lr)
mse_lr_sequential = mean_squared_error(y_test, y_pred_lr)
rmse_lr_sequential = np.sqrt(mse_lr_sequential)
r2_lr_sequential = r2_score(y_test, y_pred_lr)
print("MAE:", mae_lr_sequential)
print("RMSE:", rmse_lr_sequential)
print("R2 Score:", r2_lr_sequential)
```

```
# Calculate and print accuracy metrics for sequential Random Forest
print("\nRandom Forest Performance (Sequential):")
mae_rf_sequential = mean_absolute_error(y_test, y_pred_rf)
mse_rf_sequential = mean_squared_error(y_test, y_pred_rf)
rmse_rf_sequential = np.sqrt(mse_rf_sequential)
r2_rf_sequential = r2_score(y_test, y_pred_rf)
print("MAE:", mae_rf_sequential)
print("RMSE:", rmse_rf_sequential)
print("R2 Score:", r2_rf_sequential)
```

Its evaluation is the same as parallel

```
Linear Regression Performance (Sequential):
MAE: 4.727006291029031
RMSE: 6.1745663718829285
R2 Score: 0.7549565415424608


 Random Forest Performance (Sequential):
 MAE: 2.822545196205912
 RMSE: 4.090959301059221
 R2 Score: 0.8924326413419457
```

v) Training Time:

```
# Record training time for Linear Regression (sequential)
start_time = time.time()
lr = LinearRegression()
lr.fit(X_train, y_train)
end_time = time.time()
lr_train_time_sequential = end_time - start_time

# Record training time for Random Forest (sequential)
start_time = time.time()
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
end_time = time.time()
rf_train_time_sequential = end_time - start_time

# The fit time will be the same as the sequential version.
lr_train_time_parallel = lr_train_time_sequential

# Record training time for Random Forest (parallel)
start_time = time.time()
rf_parallel = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=n_cores)
rf_parallel.fit(X_train, y_train)
end_time = time.time()
rf_train_time_parallel = end_time - start_time
```

As noted before, scikit-learn's LinearRegression fit is not parallelizable with n_jobs. The fit time will be the same as the sequential version but it is possible in Random Forest.

```
Linear Regression Training Time (Sequential): 0.0433 seconds
Linear Regression Training Time (Parallel): 0.0433 seconds (Fit is sequential)
Random Forest Training Time (Sequential): 28.2826 seconds
Random Forest Training Time (Parallel): 19.4362 seconds
```

As we can see here, the training time for Random Forest in sequential is slower than training the model in parallel. From here we can conclude that for larger datasets or real time data, Parallel Computing will be faster and more efficient.

Overall Comparison:

To display the comparison, we will create a dataframe 'results' containing the results of the models.

```
# Create a dictionary to store the results
results = {
    'Model': ['Linear Regression', 'Linear Regression', 'Random Forest', 'Random Forest'],
    'Processing': ['Sequential', 'Parallel', 'Sequential', 'Parallel'],
    'MAE': [mae_lr_sequential, mae_lr_parallel, mae_rf_sequential, mae_rf_parallel],
    'RMSE': [rmse_lr_sequential, rmse_lr_parallel, rmse_rf_sequential, rmse_rf_parallel],
    'R2 Score': [r2_lr_sequential, r2_lr_parallel, r2_rf_sequential, r2_rf_parallel],
    'Training Time (s)': [lr_train_time_sequential, lr_train_time_parallel, rf_train_time_sequential, rf_train_time_parallel]
}

# Create a pandas DataFrame from the dictionary
results_df = pd.DataFrame(results)

# Display the DataFrame
results_df.head()
```

	Model	Processing	MAE	RMSE	R2 Score	Training Time (s)
0	Linear Regression	Sequential	4.727006	6.174566	0.754957	0.019544
1	Linear Regression	Parallel	4.727006	6.174566	0.754957	0.019544
2	Random Forest	Sequential	2.822545	4.090959	0.892433	25.270313
3	Random Forest	Parallel	2.822545	4.090959	0.892433	19.242321

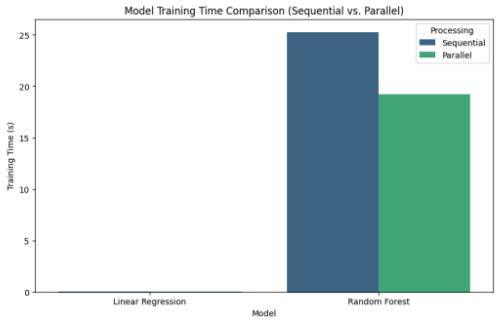
The results are same in parallel and sequential as Parallel Computing doesn't change results, it only changes speed.

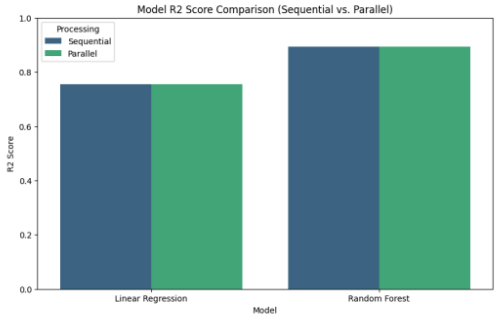
Parallel Computing decreases the time based on the amount of cores allocated on it.

Visualization:

```
# Plot Training Time Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Training Time (s)', hue='Processing', data=results_df, palette='viridis')
plt.title('Model Training Time Comparison (Sequential vs. Parallel)')
plt.ylabel('Training Time (s)')
plt.xlabel('Model')
plt.show()

# Plot R2 Score Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='R2 Score', hue='Processing', data=results_df, palette='viridis')
plt.title('Model R2 Score Comparison (Sequential vs. Parallel)')
plt.ylabel('R2 Score')
plt.xlabel('Model')
plt.ylim(0, 1)
plt.show()
```





Ethical and Professional Responsibility

Deploying an ML model has real world issues that must be addressed ethically and professionally. The problems and its solutions are addressed below:

Problem	Impact	Solution
1) If the system wrongly predicts or over predicts FRP, it can lead to unnecessary emergency messages	- Wastage of firefighting resources - Unnecessary public panic - Increases Operational Cost	- Final decision should be made by expert fire analysts and not automated entirely - Tune the model to have balanced sensitivity - Test the model on a periodic basis
2) If the model underpredicts severity, real fires may go unnoticed or unprioritized	- Loss of life - Environmental destructio and pollution - Property damage	- Use the model as a support tool and use an experience fire analyst to make the decision - Combine ML with real time data to minimize errors
3) If the model has geographic bias, since it was focused on Turkey, it might not generalize with other regions leading to unfair performance	- Areas not in training data may get wrong predictions - Some regions may be prioritized over the ones that are neglected which spreads resources unfairly	- Train on multiple regions to reduce inaccuracies - Evaluate model performance separately for each region
4) Integrating this system with data sources like drone sensors, IoT devices could raise privacy concerns	- Human activity and private property may be monitored without consent - May breach privacy	- Remove any personally identifiable information from the data - Align with national and international laws

	laws	- Inform the public about the data being used for fire detection
5) As the model gets more accurate, there is a concern that decision makers will rely on ML more than their judgement	- Experts may start to lose their skills - If the system crashes or a new fire pattern is found, they may fail to intervene in time	- Define the system as an advisor rather than the decision maker - Provide training to interpret the result and not follow it blindly

System Architecture and Deployment Planning

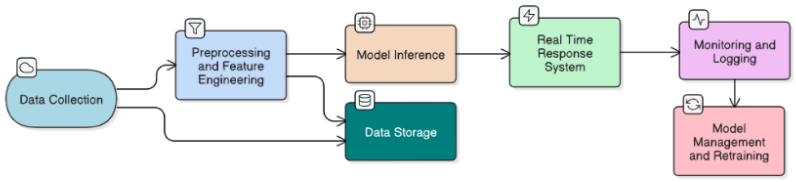


Fig: System Architecture

Data Collection Layer: This layer collects real-time wildfire data from sources such as satellites, IoT sensors, weather API data such as temperature, wind speed, etc.

Preprocessing and Feature Engineering Layer: Cleans incoming data and pre-processes data for model inference. This step will remove the outliers, fill in the missing values, aggregate brightness, etc.

Data Storage Layer: Store the new data and old data in a server.

Model Inference Layer: Prediction of the Fire Radiative Power using a pre-trained model. Models used here will be Linear Regression and Random Forest.

Real Time Response System Layer: Send alarms if the FRP is found above the critical value through the model. This can alert the firefighters and alert the public via SMS or alerting a dashboard.

Monitoring and Logging Layer: This layer monitors the reliability, model performance, and failure detection.

Model Management and Retraining Layer: This layer monitors the behavior of the model and allows continuous learning through new data

Professional Communication

Managing this project required me to think about various aspects from data preprocessing, model selection, model training, experimentation on parallel and sequential computing. I began by deciding on the problem which needed to be addressed, and I chose wildfires since wildfires can be a huge problem with how fast climate change is right now. After exploring many datasets, I found one that could be used in model training from Kaggle.

The project was done in the phases of, Data Collection, Preprocessing, Model Training and Performance Evaluation. Time management was very crucial for this project since I spent a lot of time to make sure that the code is easy to read and understand. From the parallel computing we did from PySpark and joblib, we could tell that it was much faster and efficient to train random forest on it.

There was frustration when the code would have errors, or the regression models either overfit or underfit. But overcoming them, was always satisfying and taught me how to solve such problems.

This project also helped me to find problems that may arise in the system and how we can handle it both ethically and professionally. It also taught me more about how a Machine Learning model flow works.

GitHub: https://github.com/Kaushik22864/MLPC

References

DataWithSantosh. (2025, May 7). *Advantages of PySpark in Distributed Data Processing*. Retrieved from Medium: https://medium.com/@DataWithSantosh/advantages-of-pyspark-in-distributed-data-processing-6b489275cde3

Kumar, A. (2023, November 13). *R-squared in Linear Regression Models: Concepts, Examples*. Retrieved from Analytics Yogi: https://vitalflux.com/r-squared-explained-machine-learning/

Linear Regression in Machine learning. (2025, July 11). Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/

Mean Absolute Error(MSE). (n.d.). Retrieved from encord: https://encord.com/glossary/mean-square-error-mse/

Random Forest Algorithm in Machine Learning. (2025, June 27). Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/

What is Root Mean Square Error? (n.d.). Retrieved from c3.ai: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/

ITS66604_0369834_INDVASGNMT

ORIGINALITY REPORT

18%	13%	1%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Taylor's Education Group Student Paper	6%
2	www.geeksforgeeks.org Internet Source	2%
3	quieora.ink Internet Source	2%
4	www.mdpi.com Internet Source	2%
5	deepnote.com Internet Source	1%
6	Submitted to Modern Knowledge Schools Student Paper	1%
7	Submitted to University of Western Australia Student Paper	1%
8	doc.arcgis.com Internet Source	1%
9	www.researchsquare.com Internet Source	1%
10	Submitted to Colorado Technical University Online Student Paper	<1%
11	Submitted to University of Queensland Student Paper	<1%
12	www.nature.com Internet Source	<1%
13	pt.slideshare.net	

	Internet Source	<1%
14	upcommons.upc.edu Internet Source	<1%
15	Sujith Samuel Mathew, Mohammad Amin Kuhail, Maha Hadid, Shahbano Farooq. "The Object-Oriented Approach to Problem Solving and Machine Learning with Python", CRC Press, 2025 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		