

# **TAXI FARE PREDICTION SYSTEM**

**CSCI 6444**  
**SPRING 2024**

ADITI YADAV G27474534

YOGESH MAHATO G47046199

## Table of Contents

<b>INTRODUCTION:</b> .....	3
<b>RELATED WORK</b> .....	3
1) Predictive Modeling Using Historical Data.....	3
2) Ensemble Methods for Improved Accuracy.....	4
3) Real-Time Data Processing with Big Data Tools.....	4
4) Incorporating User Feedback.....	4
<b>WORK FLOW</b> .....	5
1. DATA ACQUISITION AND PRE-PROCESSING:.....	5
2. DATA PROCESSING AND CLEANSING::.....	6
3. DISTRIBUTED DATA PROCESSING WITH PYSARKS:.....	6
4. DATA MINING TECHNIQUES AND EXPLORATORY DATA ANALYSIS (EDA).....	6
USE OF PYSARK IN DISTRIBUTED DATA PROCESSING::.....	7
<b>DATASET</b> .....	8
<b>EXPERIMENT RESULTS AND DISCUSSION</b> .....	11
INSIGHTS FROM THE DATA:.....	11
<b>CONCLUSIONS</b> .....	20
<b>REFERENCES</b> .....	23

## INTRODUCTION:

In urban landscapes, taxi services constitute a critical component of the transportation ecosystem. The burgeoning demand for efficient and reliable taxi services has catalyzed the growth of dynamic fare systems. These systems leverage vast datasets of historical taxi trip records to predict fares, enhancing transparency and trust for both service providers and consumers. This report delves into the development of a Taxi Fare Prediction System, designed to forecast taxi fares by analyzing historical data using advanced big data technologies and machine learning algorithms.

Our journey begins with the procurement of extensive taxi trip data from the New York City Taxi & Limousine Commission (TLC), a treasure trove exceeding 50GB in size. This data is not just voluminous but intricate, with details that provide insights into the operational patterns of taxis within the city. Such an endeavor demands a robust, scalable infrastructure for storage and processing, for which we turned to Amazon Web Services (AWS), leveraging the renowned S3 storage service to accommodate the heft of our dataset.

With data in hand, our odyssey continues as we navigate through the challenges of big data processing. Here, PySpark — the Python interface for Apache Spark — emerges as our cornerstone technology, offering the muscle to perform distributed data processing across a cluster of nodes. Through PySpark, we can cleanse, transform, and enrich our dataset, preparing it for the crucial phase of Machine Learning (ML) model training.

Armed with a processed and polished dataset, we embark on the crux of our system — the application of Machine Learning. By training various ML models, including the likes of Random Forest and Gradient Boosting, and evaluating them with meticulous rigor using metrics such as Root Mean Square Error (RMSE) and R-squared ( $R^2$ ), we discern the best-suited model for fare prediction.

The culmination of our efforts is manifested through the precise prediction of fares, which we then elegantly visualize using tools such as Matplotlib and Seaborn. These visualizations not only represent the outcomes but also distill complex data into comprehensible insights, ultimately empowering decision-makers with actionable intelligence.

This report encapsulates the technical narrative of our Taxi Fare Prediction System, detailing the methodologies, technologies, and analytical strategies that underpin our approach. Our intent is to shed light on the intricate interplay between big data and predictive analytics, and how they converge to drive innovation in the taxi service industry.

## RELATED WORK:

This section reviews pertinent literature on taxi fare prediction and the analysis of transportation data, highlighting the application of machine learning, data mining, and Big Data technologies. The discussed research offers valuable insights into predictive modeling and data-driven decision-making in the transportation sector.

### Predictive Modeling Using Historical Data

Wu et al. (2004) demonstrated the potential of support vector regression in predicting taxi travel times, a key component of fare calculation. Their work suggests that similar approaches could be adapted to predict fares by correlating features such as travel distance, traffic conditions, and time of day with historical fare data.

### Ensemble Methods for Improved Accuracy

A study by Weijie Wang and Yanmin Lu (2018) explored the use of ensemble learning techniques, specifically analyzing Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) in fare predictions. They emphasized the benefits of combining multiple models to enhance prediction reliability.

### Real-Time Data Processing with Big Data Tools:

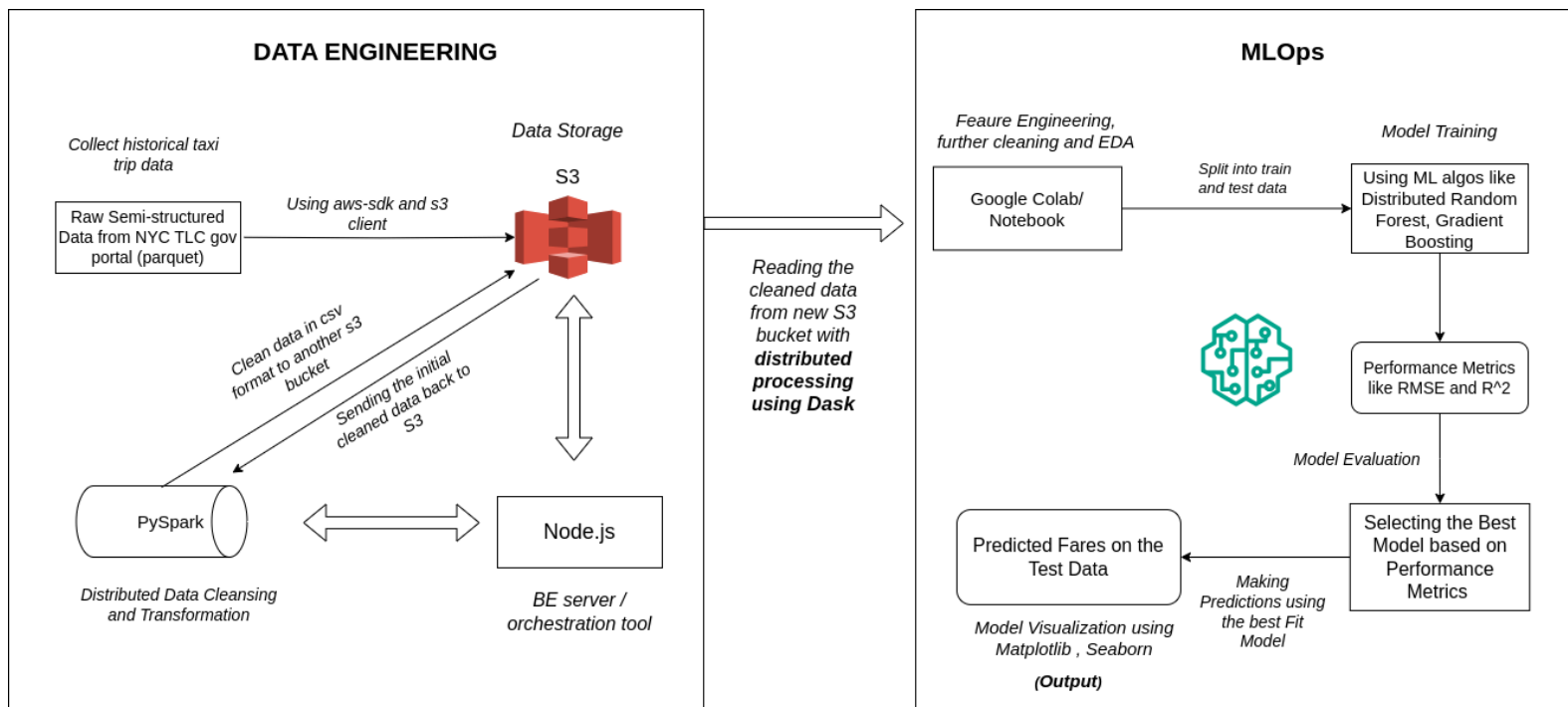
Guo et al. (2019) applied Big Data technologies, including Apache Spark and Hadoop, to process and visualize large datasets for real-time traffic and fare analysis. This approach is crucial for incorporating real-time data into fare predictions, allowing for dynamic pricing based on current traffic conditions and demand.

### Incorporating User Feedback

Wang et al. (2018) used natural language processing to analyze Yelp reviews for sentiment and text features, which could similarly be applied to taxi services by analyzing passenger feedback to predict fare satisfaction and adjust pricing strategies accordingly.

## WORKFLOW (OUR APPROACH)

In this project, we used big data technology and visualization tools in a methodical manner to evaluate the Yelp dataset. The data acquisition and pre-processing, data analysis, sentiment analysis, and data visualization are the four main steps of our approach. This section describes each stage's steps and the technology used at each one.



**Figure: System Design**

### 1. DATA ACQUISITION:

- o Data Historical taxi trip data is sourced from the official New York City Taxi & Limousine Commission (TLC) website (<https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>). This public dataset is extensive, containing over 50GB of detailed records about taxi trips.
- o The data includes crucial attributes such as pickup and dropoff locations, fare amounts, passenger counts, timestamps, trip distances, and more. These details provide a comprehensive view of taxi operations in New York City and are vital for analyzing travel patterns and fare calculations.
- o Amazon S3 is utilized to store the collected raw data. S3 is chosen for its robustness, providing high durability, maximum availability, and efficient data retrieval capabilities,

which are essential for managing large-scale datasets.

- o Data is stored in the taxi-data-raw bucket on S3. The raw data format and large volume necessitate a storage solution that can scale and handle concurrent access, making S3 an ideal choice.

## 2. DATA PROCESSING AND CLEANSING:

- o Tool: Node.js is employed as a server-side tool to orchestrate the workflow for data processing. This includes scheduling tasks, managing data flows, and handling dependencies.
- o Automation: Scripts developed in Node.js automate the initial extraction and preliminary cleaning of the dataset. These scripts facilitate the efficient management of data processing tasks, ensuring that the system can handle large volumes of data without manual intervention.

## 3. DISTRIBUTED DATA PROCESSING WITH PYSPARK

- o **PySpark**, the Python API for Apache Spark, is used in a distributed computing environment to handle intensive data processing tasks. This setup allows for parallel processing of data across multiple compute nodes, enhancing speed and efficiency.
- o Data Cleansing:
  - o Anomalies and Errors: PySpark scripts are designed to identify and remove data anomalies and incorrect entries. This ensures the accuracy and reliability of the data used in further analyses.
  - o Irrelevant Data: Filters are applied to exclude irrelevant data points that do not contribute to fare prediction, such as trips with zero passengers or no fare recorded.
- o Data Transformation:
  - o Data formats are standardized to ensure consistency across the dataset, facilitating easier manipulation and analysis. In this case, we are ultimately transforming the raw data into csv format.
  - o Most of the features in the raw are not required for our prediction system. So, we are filtering the columns that we need, and also enriching certain columns from the existing features for sending the initial cleaned data to s3 for further

processing.

#### 4. DATA MINING TECHNIQUES AND EXPLORATORY DATA ANALYSIS (EDA)

- **Data Mining Techniques:** In the pursuit of building a comprehensive predictive model, various data mining techniques are employed to uncover underlying patterns and extract actionable insights in our project.
  - **Missing Value Analysis:** To maintain the integrity of the predictive model, missing values are identified and addressed through imputation or exclusion, ensuring a complete dataset for training.
  - **Distance Calculation using Haversine Formula:** The Haversine formula is applied to calculate the spherical distance between pickup and drop-off coordinates, which is a critical factor in fare estimation.
  - **Outlier Detection and Treatment:** Identifying and handling outliers in the dataset to prevent them from skewing the model, often using techniques like Z-score or IQR (Interquartile Range) methods.
  - **Feature Extraction:** Creating new features from existing data, such as extracting the time of day from timestamps, to enhance the model's predictive capability.
- **Exploratory Data Analysis:** This analytical phase is foundational to understanding the dynamics of taxi fare pricing and involves:
  - **Visualization Techniques:** Employing charts and plots to visualize and understand the distribution of fares and the geographic spread of pickups and drop-offs.
  - **Statistical Summaries:** Providing a statistical breakdown of key variables to capture central tendencies, dispersion, and shape of the fare distribution.
  - **Correlation Studies:** Examining the relationships between different features and their correlation with fare amounts to identify significant predictors.
  - **Temporal Analysis:** Analyzing fare trends over time, including peak hours, days of the week, and seasonality, to capture temporal influences on fare prices.

In conclusion, through meticulous application of data mining techniques and comprehensive EDA, the project ensures the development of a robust, well-evaluated predictive model capable of providing accurate fare estimates for taxi trips, thereby optimizing operational efficiency and customer satisfaction.

## USE OF PYSPARK IN DISTRIBUTED DATA PROCESSING:

PySpark, the Python API for Apache Spark, is a pivotal tool in the big data ecosystem, offering robust capabilities for distributed processing. Its use in our taxi fare prediction system can be explained in the following way:

### **Distributed Computing:**

PySpark allows for distributed data processing across multiple computing nodes. This means large datasets such as the historical taxi trip data, which is inherently voluminous, can be processed in parallel, drastically reducing the time required for data cleansing, transformation, and feature engineering.

### **Scalability:**

With the increasing volume of taxi data, PySpark's ability to scale processing power horizontally is invaluable. As data grows, additional nodes can be seamlessly integrated into the processing cluster without a need for significant architectural changes.

### **Fault Tolerance:**

PySpark, running on top of the Spark framework, provides fault tolerance through its resilient distributed dataset (RDD) abstraction. Even in case of node failures, PySpark ensures that the processing tasks are re-executed on available nodes, thereby maintaining the continuity and reliability of data processing workflows.

### **Real-time Processing:**

PySpark is not only suitable for batch processing but also capable of handling real-time data streams. This feature enables the taxi fare prediction system to process real-time taxi trip data for instant insights, which is crucial for dynamic fare pricing and operational adjustments.

### **Advanced Analytics:**

PySpark supports a wide array of machine learning algorithms through its MLlib library. This makes it possible to perform sophisticated analytics, including predictive modeling, on the processed data, right within the same distributed environment, ensuring consistency and efficiency.



## Integration with Cloud Storage:

PySpark integrates seamlessly with cloud storage services like Amazon S3. This integration enables direct reading and writing of data to and from the cloud, facilitating a streamlined workflow where data can be ingested, processed, and stored back without complex data transfers or conversions.

```
def convert_and_upload(source_bucket, file_name, target_bucket):
    spark = SparkSession.builder \
        .appName("Taxi Fare Prediction Data Processing") \
        .config("spark.hadoop.fs.s3a.access.key", os.getenv('AWS_ACCESS_KEY_ID')) \
        .config("spark.hadoop.fs.s3a.secret.key", os.getenv('AWS_SECRET_ACCESS_KEY')) \
        .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
        .getOrCreate()

    parquet_path = f"s3a://{source_bucket}/{file_name}"
    df = spark.read.parquet(parquet_path)

    cleaned_df = df.selectExpr([
        "concat(tpep_pickup_datetime, '_', VendorID) as key",
        "fare_amount",
        "tpep_pickup_datetime as pickup_datetime",
        "pickup_longitude",
        "pickup_latitude",
        "dropoff_longitude",
        "dropoff_latitude",
        "passenger_count"
    ]).filter(
        (df.passenger_count > 0) &
        (df.fare_amount > 0)
    )

    # Data partitioning for optimized storage and querying
    cleaned_df = cleaned_df.repartition("pickup_datetime")

    csv_path = f"s3a://{target_bucket}/{file_name.replace('.parquet', '_clean.csv')}"
    cleaned_df.write.option("header", "true").csv(csv_path)
```

Figure: Snapshot of PySpark Process

## DATASET:

Gathered historical taxi trip data from the official New York City Taxi & Limousine Commission (TLC) website, which provides comprehensive trip records (more than 50GB) (<https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>)

Data Format (Raw Data):

## New York City TLC Trip Record Data Dictionary

Feature	Description
VendorID	A code indicating the LPEP provider that provided the record.  1 = Creative Mobile Technologies, LLC. 2 = VeriFone Inc.
lpep_pickup_datetime	The date and time when the meter was engaged.
lpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles was reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
RateCodeID	The final rate code is in effect at the end of the trip.  1 = Standard rate <b>\$3.00</b> 2 =JFK <b>bandara \$70.00</b> 3 =Newark <b>\$23.00</b> Daerah pesisir (Jauh) 4 =Nassau or Westchester <b>\$6.00</b> 5 =Negotiated fare <b>bisa di negosiasi harga</b> 6 =Group ride <b>bergrup</b>
Store_and_fwd_flag	This flag indicates whether the trip record was held in the vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server.  Y = store and forward trip N = not a store and forward trip

Payment_type	A numeric code signifying how the passenger paid for the trip.  1 = Credit card 2 = Cash 3 = No charge 4 = Dispute 5 = Unknown 6 = Voided trip
Fare_amount	The time-and-distance fare is calculated by the meter. Extra Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.
MTA_tax	\$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	\$0.30 improvement surcharge assessed on hailed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	The total amount of all tolls paid in the trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Trip_type	A code indicating whether the trip was a street hail or a dispatch that is automatically assigned based on the metered rate in use but can be altered by the driver.  1 = Street-hail 2 = Dispatch

The website has all of the data where it is stored according the years and for each year, it is stored as months from January to December

2022	
<b>January</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>February</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>March</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>April</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>May</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>June</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> </ul>	<b>July</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>August</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>September</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>October</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>November</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> <li>Green Taxi Trip Records (PARQUET)</li> <li>For-Hire Vehicle Trip Records (PARQUET)</li> <li>High Volume For-Hire Vehicle Trip Records (PARQUET)</li> </ul> <b>December</b> <ul style="list-style-type: none"> <li>Yellow Taxi Trip Records (PARQUET)</li> </ul>

## EXPERIMENT RESULTS AND DISCUSSION:

### Data Understanding

To get the best results, to get the most effective model it is really important to our data very well. Here, the given train data is a CSV file that consists of 8 features and millions of rows

Understanding the data :

```
[ ] train.head() #checking first five rows of the training dataset
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

The different variables of the data are:

**fare\_amount :** fare of the given cab ride.

**pickup\_datetime :** timestamp value explaining the time of ride start.

**pickup\_longitude :** a float value explaining longitude location of the ride start.

**pickup\_latitude:** a float value explaining latitude location of the ride start.

Variables	Description
fare_amount	Fare amount
pickup_datetime	Cab pickup date with time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers sitting in the cab

**Exploratory Data Analysis**, which includes following steps:

- Data exploration and Cleaning
- Missing values treatment
- Outlier Analysis
- Feature Selection
- Features Scaling
- Skewness and Log transformation
- Visualization
- Linear regression
- Decision Tree
- Random forest,
- Gradient Boosting
  - Random Search CV
  - Grid Search CV

**Creating some new variables from the given variables.**

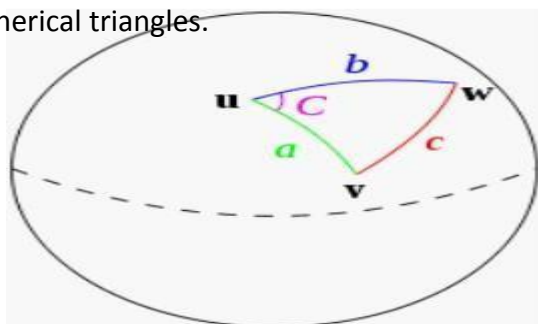
Here in our data set our variable name pickup\_datetime contains date and time for pickup.

So we tried to extract some important variables from pickup\_datetime:

- Year
- Month
- Date
- Day of Week
- Hour
- Minute

Also, we tried to find out the distance using the haversine formula which says:

The **haversine formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.



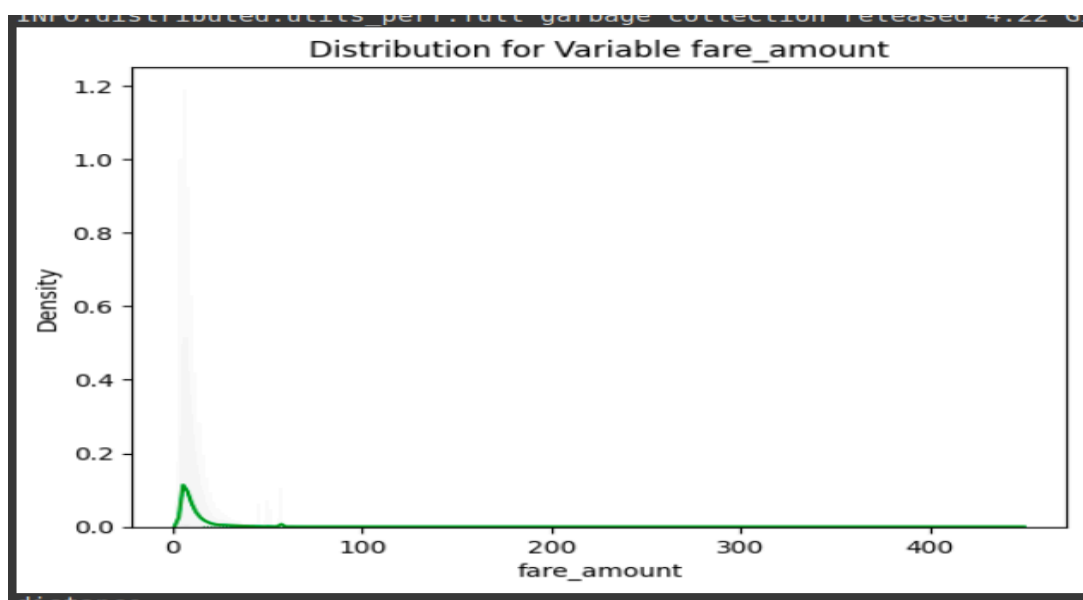
So our new extracted variables are:

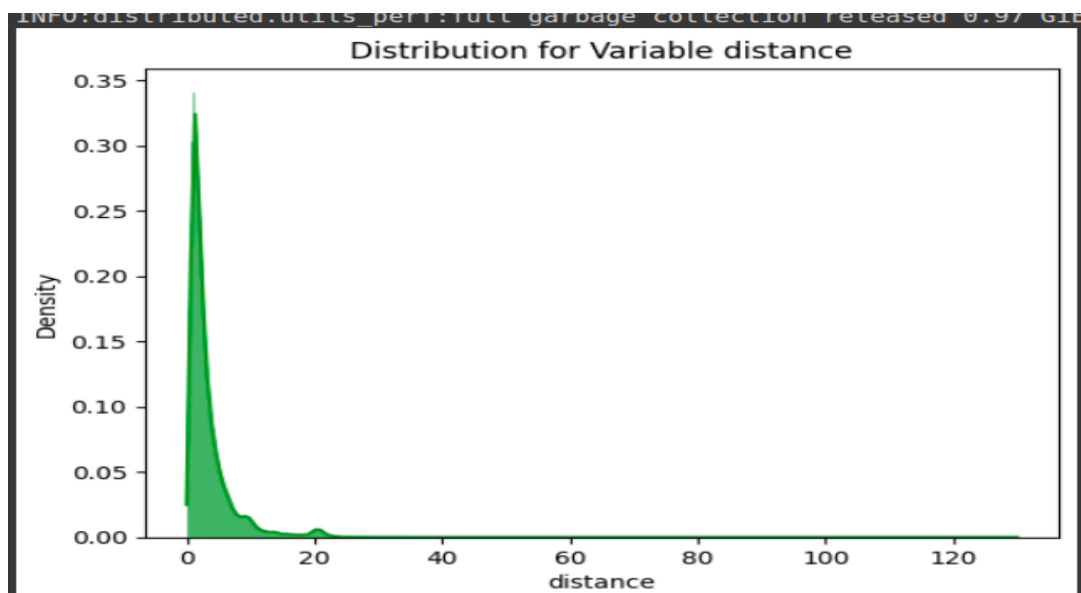
- fare\_amount
- pickup\_datetime
- pickup\_longitude
- pickup\_latitude
- dropoff\_longitude
- dropoff\_latitude
- passenger\_count
- year
- Month
- Date
- Day of Week
- Hour
- Minute
- Distance

## Feature Scaling

**Skewness** is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours is one sided skewed so by using **log transform** technique we tried to reduce the skewness of the same.

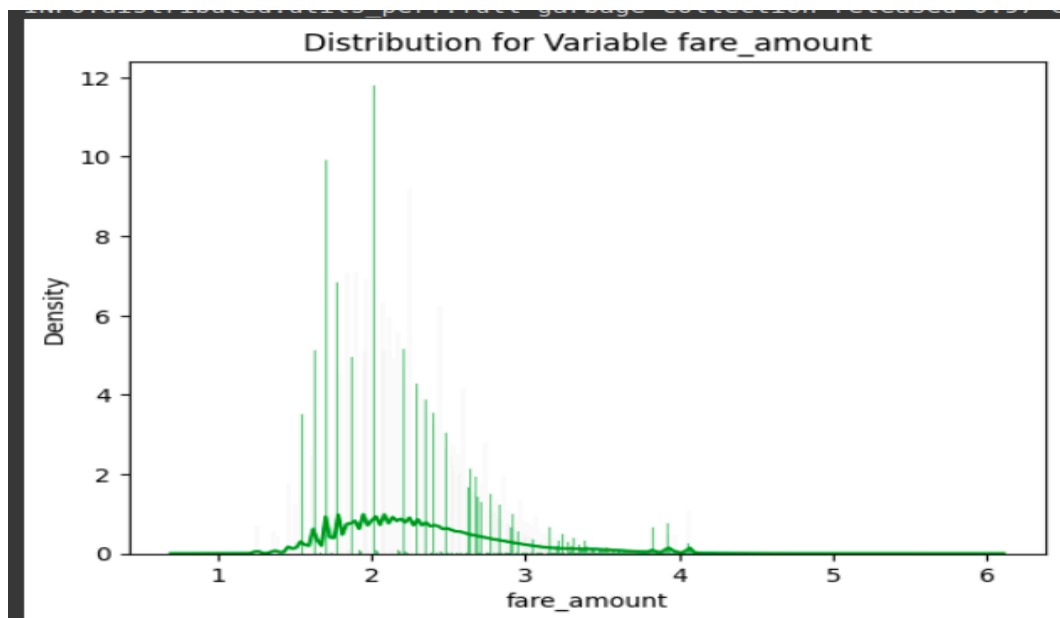
Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:

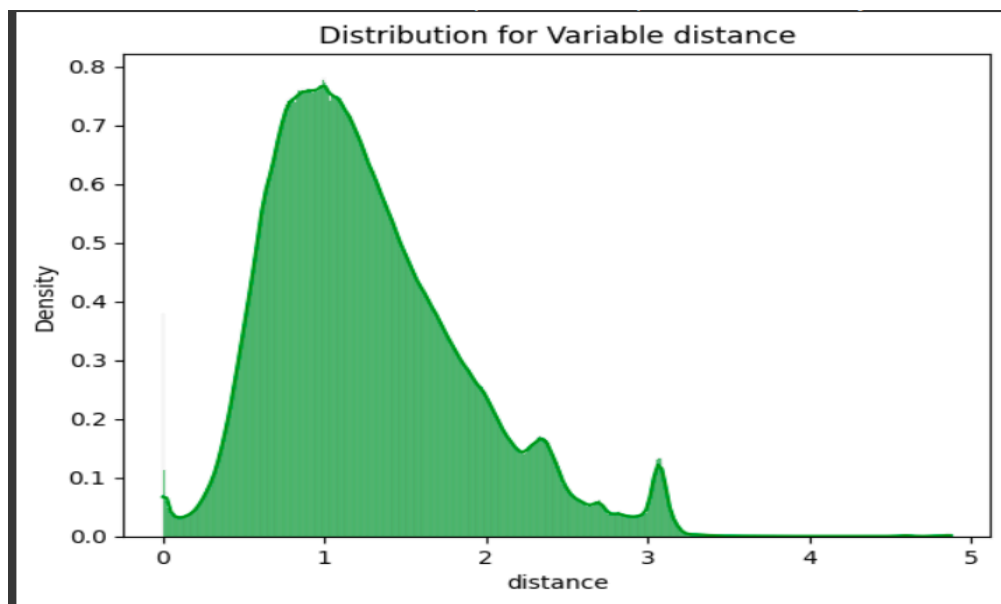




Below mentioned graphs shows the probability distribution plot to check distribution after log transformation:

As our continuous variables appear to be normally distributed, we don't need to use feature scaling techniques like normalization and standardization for the same.





### Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

Below is a screenshot of the model we build and its output:



```

##calculating RMSE for test data

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Prepare the feature matrix X by dropping the 'key' and 'fare_amount' columns
X = train_pd.drop(['key', 'fare_amount'], axis=1) # We're excluding 'key' because it's likely an identifier

# Prepare the target vector y
y = train_pd['fare_amount'].astype(float) # Converting to float to ensure numerical calculations

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the Linear Regression model
model = LinearRegression().fit(X_train, y_train)

# Predict on training and testing data
pred_train_LR = model.predict(X_train)
pred_test_LR = model.predict(X_test)

# Calculate RMSE for training and testing datasets
RMSE_train_LR = np.sqrt(mean_squared_error(y_train, pred_train_LR))
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

# Print the RMSE values
print("RMSE on Training Data:", RMSE_train_LR)
print("RMSE on Testing Data:", RMSE_test_LR)

```

RMSE on Training Data: 0.2594203198052258  
RMSE on Testing Data: 0.2594526849175961

## Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.

### Decision Tree Algorithm

```

Decision tree Model :

[ ] fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

[ ] #prediction on train data
pred_train_DT = fit_DT.predict(X_train)

#prediction on test data
pred_test_DT = fit_DT.predict(X_test)

[ ] ##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))

[ ] print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

[ ] ## R^2 calculation for train data
r2_score(y_train, pred_train_DT)

0.710640978360491

[ ] ## R^2 calculation for test data
r2_score(y_test, pred_test_DT)

```

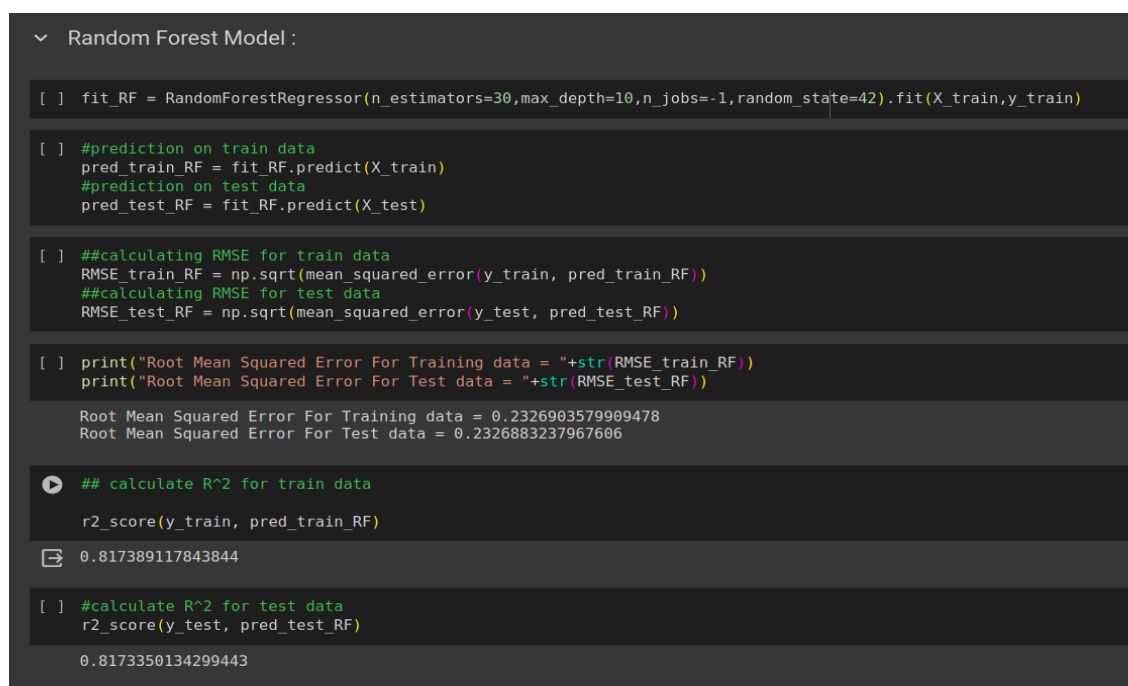
Root Mean Squared Error For Training data = 0.29290957872503126  
Root Mean Squared Error For Test data = 0.2928646227249712

## Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

**To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.**

Below is a screenshot of the model we build and its output:



```

Random Forest Model :

[ ] fit_RF = RandomForestRegressor(n_estimators=30,max_depth=10,n_jobs=-1,random_state=42).fit(X_train,y_train)

[ ] #prediction on train data
    pred_train_RF = fit_RF.predict(X_train)
    #prediction on test data
    pred_test_RF = fit_RF.predict(X_test)

[ ] ##calculating RMSE for train data
    RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
    ##calculating RMSE for test data
    RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))

[ ] print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
    print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

    Root Mean Squared Error For Training data = 0.2326903579909478
    Root Mean Squared Error For Test data = 0.2326883237967606

▶ ## calculate R^2 for train data
    r2_score(y_train, pred_train_RF)

0.817389117843844

[ ] #calculate R^2 for test data
    r2_score(y_test, pred_test_RF)

0.8173350134299443
  
```

## Hyper Parameters Tunings for optimizing the results

Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data

scientist.

Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem

**Random Search CV:** This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

Check results after using Random Search CV on Random forest and gradient boosting model.

```
# Initialize the RandomForestRegressor with a random state for reproducibility
RRF = RandomForestRegressor(random_state=0)

# Define a smaller range for n_estimators and less variation in max_depth
n_estimator = list(range(10, 51, 10)) # Reduced maximum number of estimators
depth = list(range(3, 15, 3)) # Smaller range of depths for quicker evaluations

# Create the random grid
rand_grid = {
    'n_estimators': n_estimator,
    'max_depth': depth
}

# Initialize RandomizedSearchCV with fewer iterations and CV folds for speed
randomcv_rf = RandomizedSearchCV(
    RRF,
    param_distributions=rand_grid,
    n_iter=3, # Reduced number of iterations
    cv=3, # Fewer CV folds
    random_state=0,
    n_jobs=-1 # Use all available cores
)

# Fit the model on the training data
randomcv_rf.fit(X_train, y_train)

# Predict on the test set
predictions_RRF = randomcv_rf.predict(X_test)

# Retrieve the best parameters and the best model
view_best_params_RRF = randomcv_rf.best_params_
best_model = randomcv_rf.best_estimator_

# Evaluate the model
RRF_r2 = r2_score(y_test, predictions_RRF)
RRF_rmse = np.sqrt(mean_squared_error(y_test, predictions_RRF))

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ', view_best_params_RRF)
print('R-squared = {:.2f}'.format(RRF_r2))
print('RMSE = {:.2f}'.format(RRF_rmse))
```

Random Search CV Random Forest Regressor Model Performance:  
Best Parameters = {'n\_estimators': 50, 'max\_depth': 12}  
R-squared = 0.82.  
RMSE = 0.23

We see that, out of all the models trained and evaluated based on the performance metrics like RMSE and  $R^2$ , we see that Random Search CV Random Forest Regressor Model Performance is the best fit since it has the highest  $R^2$ (0.82) and the lowest RMSE value(0.23).

## CONCLUSION

### Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions  $f(x[i,:])$  and actual outcomes  $y[i]$ .

In general, most data scientists use two methods to evaluate the performance of the model:

- I. **RMSE** (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modeled.
- II. **R Squared ( $R^2$ )**: is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains how much of the variance of the target variable is explained.
- III. We have shown both train and test data results, the main reason behind showing both the results is to check whether our data is overfitted or not.

Below table shows the model results before applying hyper tuning:

<u>Model Name</u>	<u>RMSE</u>		<u>R Squared</u>	
	<u>Train</u>	<u>Test</u>	<u>Train</u>	<u>Test</u>
`Linear Regression	0.259	0.25945	0.773	0.772
Decision Tree	0.292	0.292	0.72	0.71
Random Forest model	0.24	0.23	0.82	0.817
Gradient Boosting	0.24	0.24	0.79	0.79

Below table shows results post using hyper parameter tuning techniques:

<u>Model Name</u>	<u>Parameter</u>	<u>RMSE (Test)</u>	<u>R Squared (Test)</u>
Random Search CV	Random Forest	0.23	0.82

Above table shows the results after tuning the parameters of our two best suited models i.e. Random Forest and Gradient Boosting. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of n\_estimators, depth and CV folds.

### Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see:

- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- After this, I chose Random Forest CV to apply cross validation technique and see changes brought about by that.
- After applying tunings Random forest model shows best results compared to gradient boosting.
- So finally, we can say that the Random forest model is the best method to make predictions for this project with highest explained variance of the target variables and lowest error chances with parameter tuning technique Random Search CV.

Finally, I used this model to predict the target variable for the test data where the final output csv is sent back to another S3 bucket which is predicted\_fare\_amounts.csv

Below is the screenshot of the test\_pd.head()

```
[ ] test_pd.head()
```

pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day_of_week	distance	year	Month	Date	Day	Hour	predicted_fare_amount
40.763805	-73.981430	40.743835	1	13	1	2.323260	2015	1	27	1	13	3.578364
40.719383	-73.998886	40.739201	1	13	1	2.425353	2015	1	27	1	13	3.623484
40.751260	-73.979654	40.746139	1	11	5	0.618628	2011	10	8	5	11	1.807255
40.767807	-73.990448	40.751635	1	21	5	1.961033	2012	12	1	5	21	2.900125
40.789775	-73.988565	40.744427	1	21	5	5.387301	2012	12	1	5	21	1.892530

## REFERENCES:

[1] Panda, Gunjan, and Panda, Supriya P. "Exploratory analysis for machine learning to predict cab fares." *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, August 2019.

[2] Pravin, A., Jacob, T. P., and Asha, P. "Enhancement of plant monitoring using IoT." *International Journal of Engineering and Technology (UAE)*, vol. 7, no. 3, pp. 53-55, 2018.

[3] Bashar, Abul. "Survey of evolving deep learning neural network architectures."

[4] Jacob, T. P., Pravin, A., and Asha, P. "Arduino object follower with augmented reality." *International Journal of Engineering and Technology*, vol. 7, no. 3.27, pp. 108-110, 2018.

[5] Wu, C. H., Ho, J. M., and Lee, D. T. "Travel-time prediction using support vector regression." *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 4, pp. 276-281, 2004.

[6] Van Lint, J. W. C., Hoogendoorn, S. P., and van Zuylen, Henk J. "Freeway travel time prediction with state-space neural networks under missing data." *Transportation Research Part C: Emerging Technologies*, vol. 13, no. 5, pp. 347-369, 2005.

[7] Vanajakshi, L., Subramanian, S. C., and Sivanandan, R. "Travel time prediction under heterogeneous traffic conditions using GPS data from buses." *IET Intelligent Transport Systems*, vol. 3, no. 1, pp. 1-9, 2009.

[8] Kanimozhi, V., and Jacob, T. P. "Calibration of machine learning classifiers in network intrusion detection using cloud computing." *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 6, 2019.

[9] Wang, Weijie, and Lu, Yanmin. "Analysis of MAE and RMSE in rounding model assessment." *ICMEMSCE, IOP Publishing*, vol. 324, 2018, DOI: 10.1088/1757-899X/324/1/012049.

[10] Jacob, T. P., and Ravi, T. "Optimal regression test case prioritization using genetic algorithms." *Life Science Journal*, vol. 10, no. 3, pp. 1021-1033, 2013.

[11] Qian, X., and Ukkusuri, S. V. "Time-of-Day pricing in taxi markets." *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, June 2017.

[12] Jacob, T. P., and Ravi, T. "Technique for reducing regression test effort." *Indian Journal of Science and Technology*, vol. 6, no. 8, pp. 5065-9, 2013.

[13] Yildirimoglu, Mehmet, and Geroliminis, Nikolas. "Experienced travel time prediction for congested freeways." *Transportation Research Part B: Methodological*, vol. 53, pp. 45-63, 2013.

[14] Hinduja, D., Dheebhika, R., and Jacob, T. P. "Enhanced character recognition using deep neural networks." *2019 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0438-0440, IEEE, April 2019.

[15] Manoharan, J. Samuel. "Study of extreme learning machine variants and performance on classification algorithms." *Journal of Soft Computing Paradigm (JSCP)*, vol. 3, no. 02, pp. 83-95, 2021.