PHASE 5 SUBMISSION DOCUMENT

Project Title: TRAFFIC MANAGEMENT

Phase 5: *Project Documentation & Submission*

Topic: In this section, I will document the complete project and prepare it for submission.

Introduction:

The growing urbanization and increasing population in cities around the world have put immense pressure on transportation systems. Traffic congestion, air pollution, and accidents have become common challenges that cities face. To address these issues and create more efficient and sustainable transportation networks, cities are turning to emerging technologies, with the Internet of Things (IoT) at the forefront of this transformation. In this extensive exploration, we will delve into the significant role that IoT plays in traffic management, how it's changing the urban landscape, its associated benefits, challenges, and the potential future developments in this field.

IoT in Traffic Management: A Game Changer

IoT in traffic management involves the deployment of a network of interconnected sensors, cameras, and data processing devices throughout the urban infrastructure. These devices collect real-time data on various aspects of traffic flow, including vehicle movement, pedestrian activity, weather conditions, and road infrastructure status. The data collected is then analyzed and utilized to make informed decisions about traffic control, optimize road networks, and improve the overall urban mobility experience.

Benefits of IoT in Traffic Management:

1. Real-Time Monitoring: IoT enables real-time traffic monitoring, allowing authorities to promptly respond to accidents, congestion, or adverse weather conditions. This helps reduce traffic bottlenecks and enhance safety.

2. Smart Traffic Signals: Adaptive traffic signal systems adjust signal timings based on real-time traffic conditions, reducing idle time and minimizing congestion.

3. Data-Driven Decision Making: Traffic data collected through IoT is invaluable for urban planning. It informs decisions about road expansion, maintenance, and the development of public transit systems.

4. Environmental Benefits: By optimizing traffic flow and reducing congestion, IoT contributes to reduced fuel consumption and lower greenhouse gas emissions, making cities more environmentally friendly.

Challenges and Considerations:

1. Privacy and Security: Collecting and processing vast amounts of traffic data can raise privacy concerns. Ensuring data security and protecting the anonymity of individuals is a critical challenge.

2. Data Overload: Handling and analyzing the massive volumes of data generated by IoT devices can be overwhelming. Cities need robust infrastructure and data processing capabilities to manage this information effectively.**3. Integration and Standards:** Different cities and regions may use different technologies and standards, making interoperability and data sharing a challenge.**4. Cost and**

Infrastructure: Implementing IoT systems requires significant investment in infrastructure and technology. Funding can be a barrier for some cities.

Future of IoT in Traffic Management:

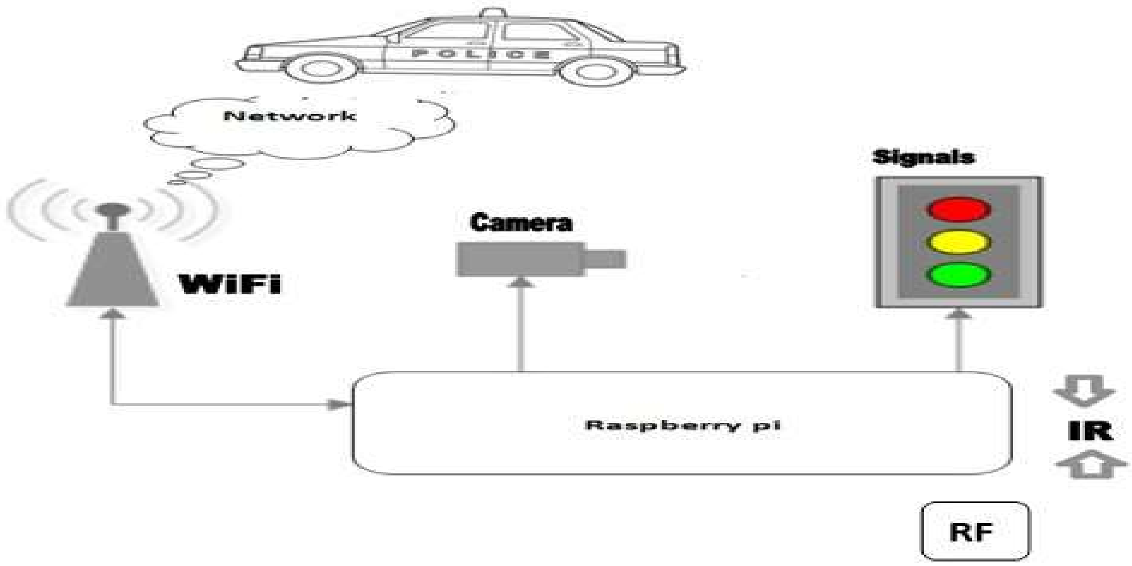
The potential for IoT in traffic management is vast and ever-evolving. Here are some future developments to watch for:

1. Autonomous Vehicles: The integration of IoT will be crucial for the success of autonomous vehicles. IoT can assist in vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication, ensuring safer and more efficient transportation.

2. Predictive Analytics: Advanced data analytics, combined with AI, will enable cities to predict traffic patterns, making traffic management more proactive than reactive.

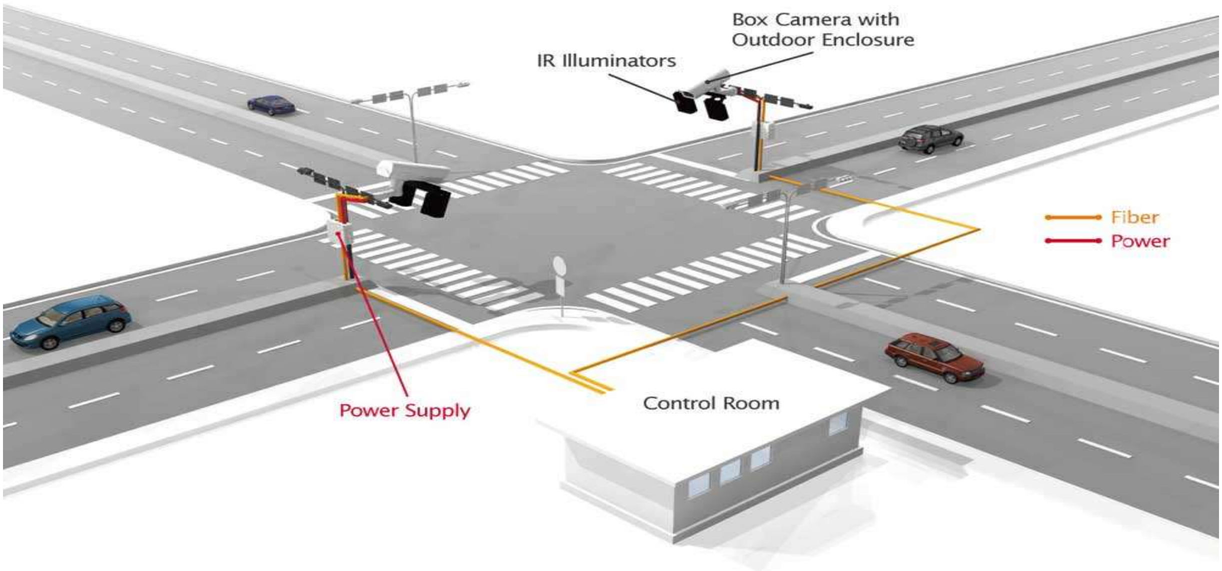
3. Smart Parking: IoT can be used to develop smart parking solutions, helping drivers find parking spaces more easily and reducing traffic congestion caused by the search for parking.

4. Sustainability: As cities prioritize sustainability, IoT can help optimize traffic flow in a way that reduces emissions and promotes cleaner transportation options, such as electric vehicles.



Coding for historical traffic data:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Load your historical traffic data CSV file (replace 'historical_traffic_data.csv'
with your file path)
data = pd.read_csv('historical_traffic_data.csv')

# Split the data into features (X) and the target variable (y)
X = data[['Hour', 'DayOfWeek', 'WeatherCondition', 'RoadCondition']]
y = data['CongestionLevel']

# Convert categorical variables into numerical features (one-hot encoding)
X = pd.get_dummies(X, columns=['WeatherCondition', 'RoadCondition'],
drop_first=True)

# 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)

# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
```



In this section begin building your project by loading and preprocessing the dataset: Data Processing: Understand the Current Situation

1. Import Datasets

Before starting to think about the Optimization Model, your priority is to understand the current situation.

Starting with unstructured data coming from several sources, we'll need to build a set of data frames to model our network and provide visibility on the loading rate and list of stores delivered for each

Records of Deliveries per Store

Deliveries Records

`Delveries_record_ccv`

Date	Truck_ID	Store_ID	FTL	Order	BOX	SKU	Loading (Tons)
9/1/2016	Truck_ID1	Store_ID1	3.5	16	311	83	2.404

Date	Truck_ID	Store_ID	FTL	Order	BOX	SKU	Loading (Tons)
9/1/2016	Truck_ID1	Store_ID2	3.5	18	178	83	1.668
9/1/2016	Truck_ID2	Store_ID3	3.5	10	74	54	0.81
9/1/2016	Truck_ID2	Store_ID4	3.5	19	216	88	2.413
9/1/2016	Truck_ID3	Store_ID5	3.5	10	117	54	1.119
9/1/2016	Truck_ID3	Store_ID6	3.5	15	294	92	2.962
9/1/2016	Truck_ID4	Store_ID7	3.5	5	42	19	0.421
9/1/2016	Truck_ID4	Store_ID8	3.5	12	125	88	1.138
9/1/2016	Truck_ID5	Store_ID9	5	18	201	95	2.19

Store Address

Store_address.csv

Search this file…

Code	city	Long	Lat	address
Store_ID1	City_Store1	31.952792	118.8192708	Address_1
Store_ID2	City_Store2	31.952792	118.8192718	Address_2
Store_ID3	City_Store3	31.675948	120.7468221	Address_3
Store_ID4	City_Store4	31.664448	120.7700006	Address_4
Store_ID5	City_Store5	31.750971	119.9478857	Address_5
Store_ID6	City_Store6	31.791351	119.9232302	Address_6
Store_ID7	City_Store7	31.79233	119.9768294	Address_7
Store_ID8	City_Store8	31.982972	119.5832084	Address_8

Store_ID9	City_Store9	31.996161	119.6341775	Address_9
Store_ID10	City_Store10	31.885547	121.1886473	Address_10
Store_ID11	City_Store11	30.310079	120.1515734	Address_11
Store_ID12	City_Store12	31.383616	121.2569408	Address_12
Store_ID13	City_Store13	31.387863	121.2797154	Address_13

Transportation Costs

Transportation_cost.csv

Search this file…

City_En	3.5T (Rmb)	5T (Rmb)	8T (Rmb)	3.5T (Rmb/Ton)	5T (Rmb/Ton)	8T (Rmb/Ton)
City_1	485	650	800	139	130	100
City_2	640	700	820	183	140	103
City_3	690	780	890	197	156	111
City_4	810	1,000	1,150	231	200	144
City_5	1,300	1,568	1,723	371	314	215
City_6	1,498	1,900	2,100	428	380	263
City_7	980	1,250	1,450	280	250	181
City_8	1,350	1,450	1,500	386	290	188
City_9	1,350	1,450	1,500	386	290	188
City_10	850	1,000	1,200	243	200	150

2. Listing of stores delivered by each route

Let us process the initial data frame to list all stores delivered for each route.

1 Route = 1 Truck ID + 1 Date

Create Transport Plan

```
Def transport_plan(data, dict_trucks, capacity_dict):
    # List of Stores per Truck for each DAY
    Df_plan = pd.DataFrame(data.groupby([‘Date’, ‘TruckID’])[‘Code’].apply(list))
    Df_plan.columns = [‘List_Code’]
    # List of Box Quantity
    Df_plan[‘List_BOX’] = data.groupby([‘Date’, ‘TruckID’])[‘BOX’].apply(list)
    # Mean of FTL
    Df_plan[‘FTL’]  = data.groupby([‘Date’, ‘TruckID’])[‘FTL’].mean()
    Df_plan[‘Capacity(T)’] = df_plan[‘FTL’].map(capacity_dict)
    Df_plan[‘List_Loading’] = data.groupby([‘Date’, ‘TruckID’])[‘Loading(T)’].apply(list)
    Df_plan[‘Count’] = df_plan[‘List_Loading’].apply(lambda t: len(t))
    Df_plan[‘Total_tons(T)’] = data.groupby([‘Date’, ‘TruckID’])[‘Loading(T)’].sum()

    # Distribute: one shipment per col
    # Stores
    D = df_plan[‘List_Code’].apply(pd.Series)
    For col in d:
       Df_plan[“Store%d” % (col+1)] = d[col]
    # Boxes number
    D = df_plan[‘List_BOX’].apply(pd.Series)
    For col in d:
       Df_plan[“Box%d” % (col+1)] = d[col]
    # Shipments Tonnage
```

```
D = df_plan['List_Loading'].apply(pd.Series)
For col in d:
    Df_plan["Tons%d" % (col+1)] = d[col]

# Fill NaN + Drop useless columns
Df_plan.fillna(0, inplace = True)
If 1 == 0:
        Df_plan.drop(['List_Code'], axis = 1, inplace = True)
        Df_plan.drop(['List_BOX'], axis = 1, inplace = True)
        Df_plan.drop(['List_Loading'], axis = 1, inplace = True)

Return df_plan
```

Example Transport Plan

Transport_plan.csv

Search this file…

Date	TruckID	List_Code	Capacity(T)	List_Loading	Count	Total_tons(T)	Store1	Store2	Store3	Store4	Box1	Box2	Box3	Box4	Tons1	Tons2	Tons3	Tons4	Occupation(%)	Available(T)
9/1/2016	Truck_ID1	[‘Store_ID6’]	3.5	[2.91]	1	2.91	ID6	0	0	0	243	0	0	0	2.91	0	0	0	83.14	0.59
9/1/2016	Truck_ID2	[‘Store_ID34’, ‘Store_ID22’, ‘Store_ID9’]	3.5	[0.3, 1.37, 0.47]	3	2.14	ID34	ID22	ID9	0	31	116	44	0	0.3	1.37	0.47	0	61.14	1.36
9/1/2016	Truck_ID3	[‘Store_ID18’]	3.5	[1.5]	1	1.5	ID18	0	0	0	174	0	0	0	1.5	0	0	0	42.86	2
9/1/2016	Truck_ID4	[‘Store_ID37’]	3.5	[2.3]	1	2.3	ID37	0	0	0	179	0	0	0	2.3	0	0	0	65.71	1.2

9/1/2016 Truck_ID5 ['Store_ID34', 'Store_ID48'] 3.5 [2.14, 0.51] 2 2.65 ID34 ID48 0 0 168 46 0 0 2.14 0.51 0 0 75.71 0.85

Add cities covered by each route

Let us now calculate Transportation Costs invoiced by carriers for each route:

```
## Pricing Functions
Def f_maxcity(list_cities, list_price):
        Return list_cities[list_price.index(max(list_price))] # Index of Maximum Price
Def inner_stops(list_cities, max_city):
        Return list_cities.count(max_city) – 1
Def outer_stops(list_cities, max_city):
        Return len(list_cities) – (list_cities.count(max_city))
Def total_price(max_price, inner_stops, outer_stops, inner_price, outer_price):
        Return max_price + inner_stops * inner_price + outer_stops * outer_price

# Calculate Price
Def plan_price(df_strinfo, df_plan, inner_price, outer_price):

        # Dictionnary Ville
        Dict_ville = dict(zip(df_strinfo.Code.values, df_strinfo.City.values))

        # Price per Truck Size： 3.5T, 5T, 8T
        Dict_35, dict_5, dict_8 = [dict(zip(df_strinfo.City.values, df_strinfo[col].values)) for col in ['3.5T', '5T', '8T']]
```

```
# Mapping Cities
F_ville = lambda t: [dict_ville[i] for I in t] # literal_eval(t)

# Mapping Price
F_35 = lambda t: [dict_35[i] for I in t]
F_5 = lambda t: [dict_5[i] for I in t]
F_8 = lambda t: [dict_8[i] for I in t]

# Mapping Price
Df_plan['List_City'] = df_plan['List_Code'].map(f_ville)
Df_plan['List_Price35'] = df_plan['List_City'].map(f_35)
Df_plan['List_Price5'] = df_plan['List_City'].map(f_5)
Df_plan['List_Price8'] = df_plan['List_City'].map(f_8)

# Maximum Price City
F_maxprice = lambda t: max(t) # Maximum Price

# Mapping First City
Df_plan['Max_Price35'] = df_plan['List_Price35'].map(f_maxprice)
Df_plan['Max_Price5'] = df_plan['List_Price5'].map(f_maxprice)
Df_plan['Max_Price8'] = df_plan['List_Price8'].map(f_maxprice)
Df_plan['Max_City'] = df_plan.apply(lambda x: f_maxcity(x.List_City, x.List_Price35), axis = 1)

# Inner City Stop
```

Df_plan['Inner_Stops'] = df_plan.apply(lambda x: inner_stops(x.List_City, x.Max_City), axis = 1)

Df_plan['Outer_Stops'] = df_plan.apply(lambda x: outer_stops(x.List_City, x.Max_City), axis = 1)

Total Price

Df_plan['Price35'] = df_plan.apply(lambda x: total_price(x.Max_Price35, x.Inner_Stops, x.Outer_Stops,

Inner_price, outer_price), axis = 1)

Df_plan['Price5'] = df_plan.apply(lambda x: total_price(x.Max_Price5, x.Inner_Stops, x.Outer_Stops,

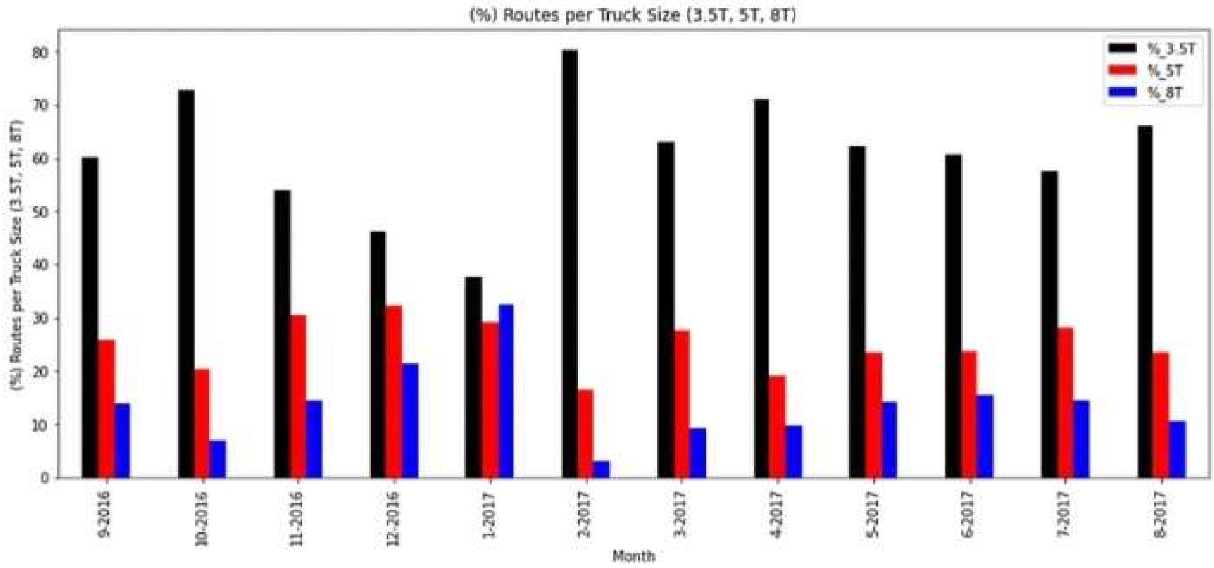
Inner_price, outer_price), axis = 1)

Df_plan['Price8'] = df_plan.apply(lambda x: total_price(x.Max_Price8, x.Inner_Stops, x.Outer_Stops,

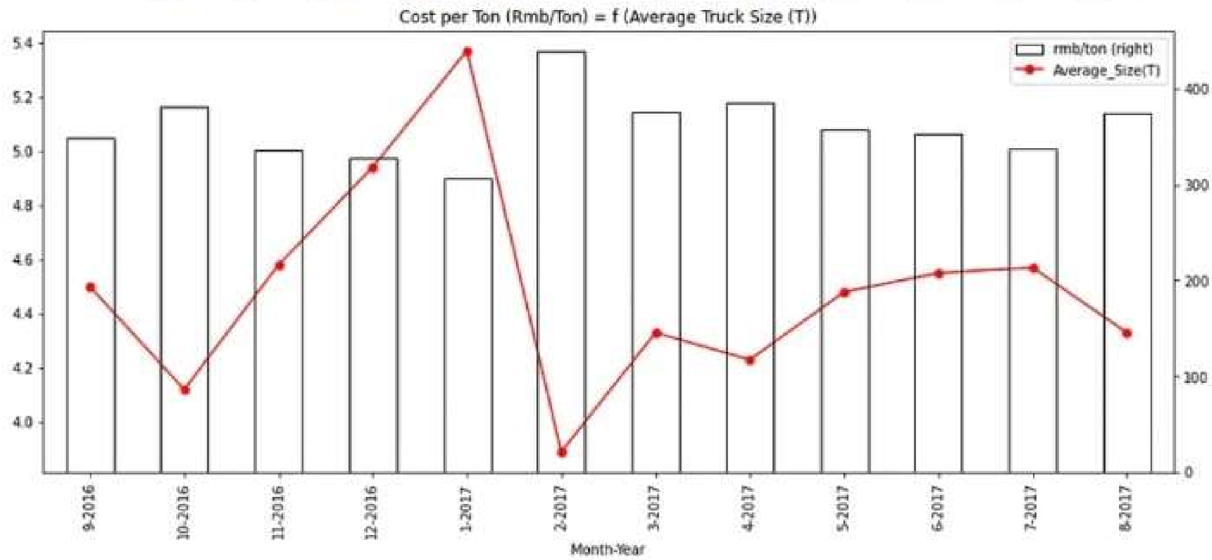
Inner_price, outer_price), axis = 1)

Return df_plan

Visualization: % Deliveries per Truck Size



(%) of Route per Truck Size (3.5T, 5T, 8T) — (Image by Author)



Impact of Average Truck Size (Ton) on Overall Cost per Ton (Rmb/Ton) — (Image by Author)

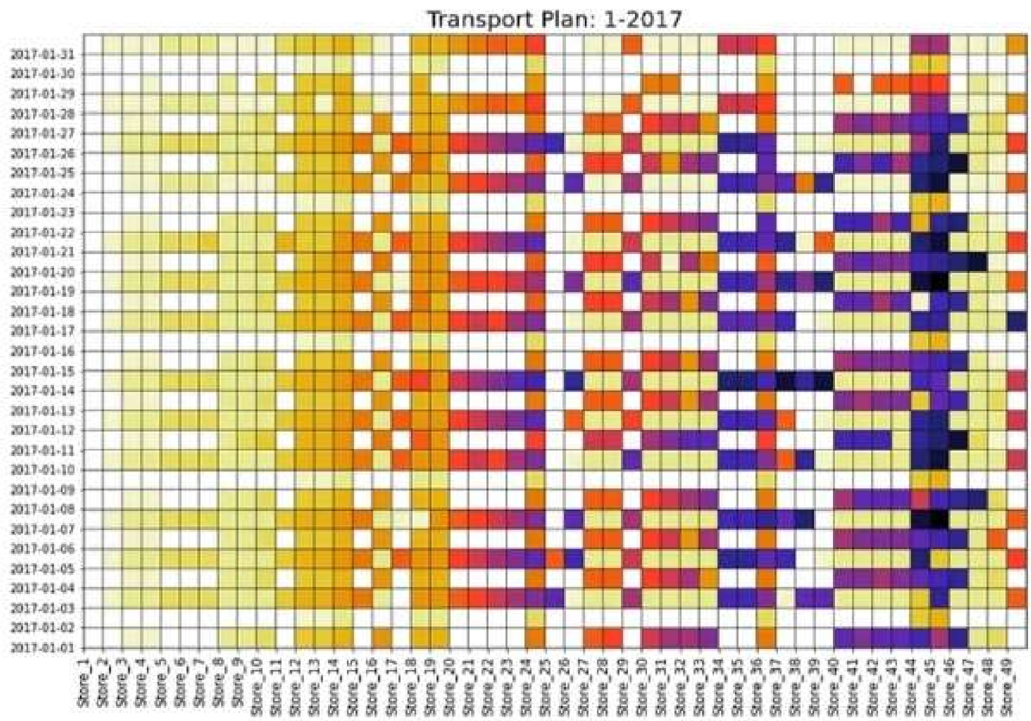
Insights

- **Average Truck Size:** a large majority of small trucks
- **Cost per ton:** the inverse proportion of cost per ton and average truck size

Understand Current Situation: Visualization

1. Transportation Plan Visualization

Objective: Get a simple visualization of all deliveries per day with a focus on the number of different routes.



Transportation Plan: January 2017 — (Image by Author)

Solution: Python's Matplotlib grid function

- **Columns:** 1 Column = 1 Store
- **Rows:** 1 Row = 1 Day
- **Colour = White:** 0 delivery
- **Colours:** 1 Color = 1 Route (1 Truck)

Geographical Visualization of Store Deliveries

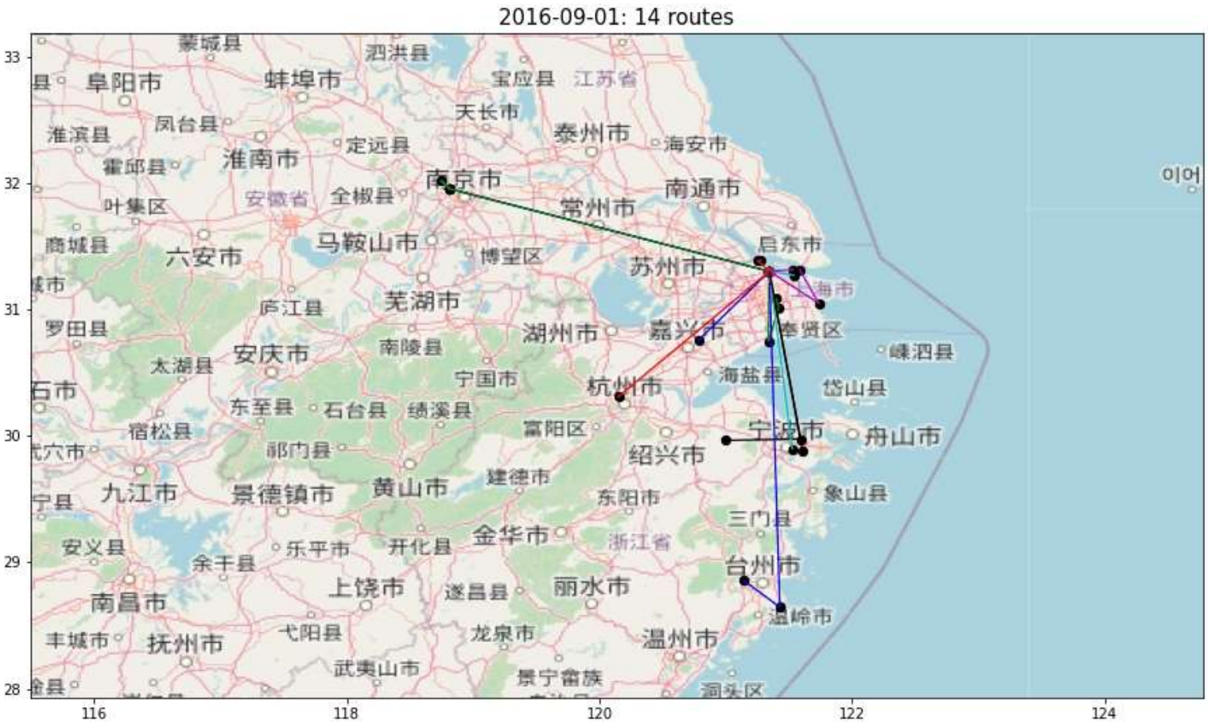
Objective

Visualisation of geographical locations delivered in the same route

Solution

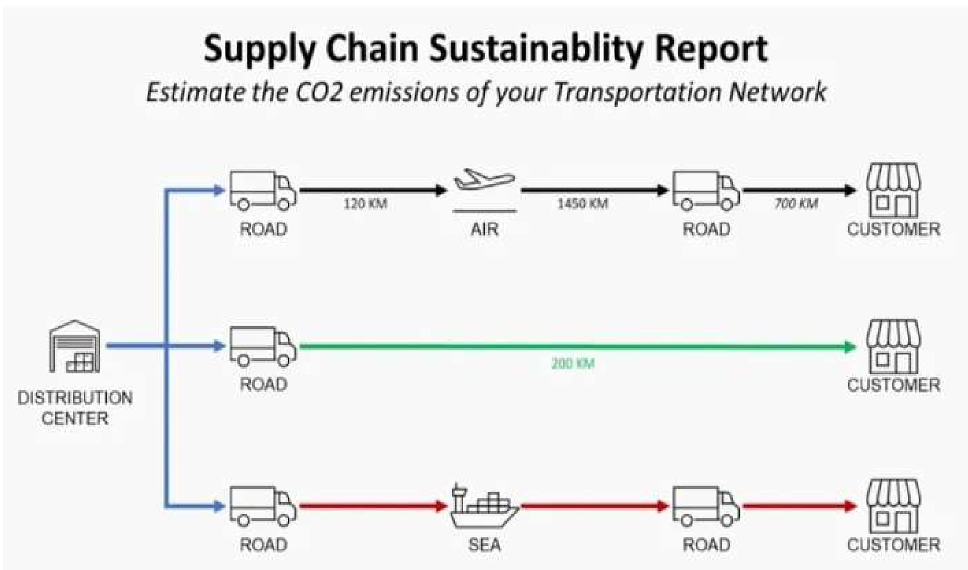
OpenStreet Map + Matplotlib Scatter Plot

Visualization of the different routes covered per day



Next Steps

1. **Measure the Environmental Impact**



In addition to cost reduction, you can also target CO_2 Emissions reductions by Optimizing your Transportation Network.

Routing Optimization: Number of Deliveries per Route

Dataframe with historical records processed

Current transportation plan

A model to calculate transportation cost per route based on cities delivered

Visualisation of the number of different routes per day

Visualisation of geographical locations delivered per Route

Next steps are

Routing: increase the number of stores delivered for each route

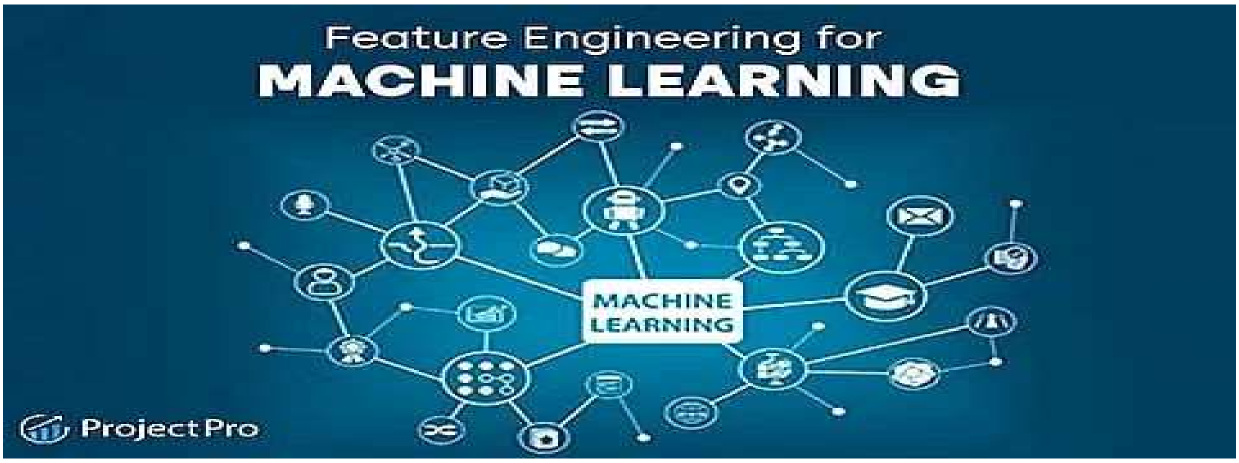
Fleet Allocation: ensure uniform workload distribution

Delivery Frequency: reduce the number of deliveries per week to increase the quantity per shipment

Simulate Impact: savings we can get from optimization listed above

In this section continue building the project by performing different activities like feature engineering, model training, evaluation etc. as per the instructions in the project:

Feature Engineering:



Traffic management project in IoT with a focus on feature engineering. Feature engineering is a crucial step in data preparation that involves selecting, transforming, and creating relevant features to improve the performance of your IoT traffic management system. Here's a step-by-step guide on how to proceed:

1. Data Collection: Start by gathering data from various IoT sensors and devices deployed in the traffic management system. These sensors can include cameras, vehicle detectors, environmental sensors (for weather conditions), and GPS devices on vehicles.

2. Data Preprocessing:

- Data Cleaning: Clean the collected data by handling missing values, outliers, and noise in the sensor data.

- Data Integration: Combine data from different sensors and sources, aligning timestamps and ensuring data consistency.

-Data Normalization/Scaling: Normalize or scale data to bring all features to a common scale.

3. Feature Selection:

- Identify which features are relevant for traffic management. For instance, speed, vehicle count, vehicle type, weather conditions, road conditions, and traffic signals.

- Use techniques like correlation analysis to determine which features have a strong impact on traffic flow and congestion.

4. Feature Engineering:

- Time-Based Features: Create time-related features such as hour of the day, day of the week, or time since the last traffic update. These features can help capture temporal patterns.

- Spatial Features: Compute spatial features like distance to the nearest traffic signal, congestion, or accident.

- Aggregated Features: Calculate aggregated statistics over a specific time window, such as average speed, traffic density, or traffic flow rate.

- Derived Features: Create features that are derived from existing data, such as acceleration, deceleration, and queue length.

- Categorical Features: If you have categorical data, encode it into numerical features using techniques like one-hot encoding.

5. Dimensionality Reduction:

- If your dataset has too many features, consider using dimensionality reduction techniques like Principal Component Analysis (PCA) to reduce the number of features while preserving essential information.

6. Feature Testing and Validation:

- Split your dataset into training, validation, and testing sets to assess the performance of different feature sets.

- Use machine learning models to evaluate how well different features impact traffic prediction and management.

7. Iterate and Refine:

- Continuously iterate on the feature engineering process. Experiment with different combinations of features and transformations to improve the accuracy of traffic predictions.

8. Model Building:

- Once you've finalized your feature set, build machine learning or deep learning models to predict traffic conditions, congestion, and optimize traffic management.

9. Real-Time Integration:

- Implement the system to process real-time data from IoT sensors and update traffic management strategies accordingly. This may involve deploying the system on edge devices or cloud infrastructure.

10. Monitoring and Maintenance:

- Regularly monitor the system's performance and make adjustments as needed to adapt to changing traffic conditions or sensor data quality.

Feature engineering is an ongoing process that can significantly impact the effectiveness of your IoT traffic management system. It requires domain knowledge and experimentation to fine-tune the feature set for optimal results.

Example:

```
import pandas as pd


# Sample traffic data as a Pandas DataFrame
```

```
data = pd.DataFrame({
    'timestamp': ['2023-10-25 08:00:00', '2023-10-25 08:15:00', '2023-10-25
08:30:00'],
    'speed': [45, 40, 35],
    'vehicle_count': [50, 45, 40]
})

# Convert the 'timestamp' column to a datetime object
data['timestamp'] = pd.to_datetime(data['timestamp'])

# Feature Engineering
# Time-Based Features
data['hour_of_day'] = data['timestamp'].dt.hour
data['day_of_week'] = data['timestamp'].dt.dayofweek
data['time_since_last_update'] = data['timestamp'].diff().dt.total_seconds()

# Spatial Features (dummy data for demonstration)
data['distance_to_traffic_signal'] = [100, 150, 200]
data['congestion'] = [0.1, 0.2, 0.3]

# Display the updated DataFrame
print(data)
```

Model training:

Model training in the context of traffic management using IoT. Model training is a crucial component of building an effective traffic management system, as it enables the system to make predictions and decisions based on the data collected from IoT devices and sensors. Here's an overview:

1. Data Preparation:

- **Data Collection**: Gather data from IoT devices and sensors deployed in the traffic management system. This data may include information on vehicle speed, count, type, environmental conditions, and other relevant parameters.

- **Data Preprocessing**: Clean and preprocess the data to handle missing values, outliers, and noise. Ensure that the data is in a suitable format for training a machine learning model.

- **Feature Engineering**: As discussed previously, feature engineering involves selecting, transforming, and creating relevant features from the raw data. These features capture important information and patterns for traffic prediction and control.

2. Data Splitting:

- Divide the dataset into training, validation, and testing sets. The training set is used to train the machine learning model, the validation set helps in hyper parameter tuning, and the testing set is used to evaluate the model's performance.

3. Model Selection:

- Choose an appropriate machine learning or deep learning model for your traffic management task. Common models include linear regression, decision trees, random forests, neural networks, or more advanced methods like recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

4. Model Training:

- Train the selected model on the training data. During training, the model learns the underlying patterns and relationships in the data. The goal is to minimize the difference between the model's predictions and the actual traffic conditions.

- **Optimization**: Define an appropriate loss function and use optimization techniques like gradient descent to update the model's parameters.

5. Hyper parameter Tuning:

- Experiment with different hyper parameters of the model to find the best configuration. This process is typically done using the validation set. Hyper parameters include learning rates, the number of layers in neural networks, and regularization terms.

Example:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset with engineered features (replace with your data)
data = pd.read_csv("traffic_data_with_features.csv")
```

```
# Split the data into features (X) and target (y)
X = data[['hour_of_day', 'day_of_week', 'time_since_last_update',
'distance_to_traffic_signal', 'congestion']]
y = data['speed']  # The target variable we want to predict, e.g., traffic speed

# 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)

# Initialize and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

Evaluation:

Evaluating the performance of your traffic management system in IoT is a critical step to ensure that it meets the desired objectives and provides accurate predictions and decisions. Evaluation helps identify strengths and weaknesses, refine the system, and validate its effectiveness. Here's how you can perform evaluation in traffic management using IoT:

1. Define Evaluation Metrics:

- Start by defining the evaluation metrics that align with your project's goals. Common metrics for traffic management include:

  - **Mean Squared Error (MSE)**: Measures the average squared difference between predicted and actual values (e.g., traffic speed).

  - **Root Mean Squared Error (RMSE):** The square root of MSE, providing a more interpretable scale.

  - **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and actual values.

  - **R-squared (R2):** Indicates the proportion of variance in the data explained by the model. Higher R2 values suggest better model performance.

  - **Precision, Recall, and F1-Score:** If your system includes classification tasks (e.g., accident detection), consider these metrics for binary classification.

2. Data Splitting:

- Split your data into training, validation, and testing sets. The validation set is used for hyper parameter tuning, while the testing set is reserved for the final model evaluation.

3. Model Evaluation:

- Evaluate the performance of your traffic prediction or decision-making models on the testing set using the predefined evaluation metrics. For regression tasks, you can calculate MSE, RMSE, MAE, and R2 using the model's predictions and the actual data. For classification tasks, use precision, recall, and F1-score.

- If your traffic management system involves multiple tasks (e.g., predicting traffic speed and detecting accidents), evaluate each task separately.

4. Real-World Testing:

- If possible, perform real-world testing to assess how well the system performs in actual traffic conditions. This may involve pilot deployments or controlled experiments in real traffic scenarios.

5. Compare Against Baselines:

- Compare your system's performance against baseline models or existing systems. This helps establish the added value of your IoT-based traffic management system.

Example:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from math import sqrt

# Load the dataset with engineered features (replace with your data)
```

```
data = pd.read_csv("traffic_data_with_features.csv")

# Split the data into features (X) and target (y)
X = data[['hour_of_day', 'day_of_week', 'time_since_last_update',
'distance_to_traffic_signal', 'congestion']]
y = data['speed']  # The target variable we want to predict, e.g., traffic speed

# 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)

# Initialize and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse}")
```

```
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2): {r2}")
```

conclusion:

Traffic management using IoT leverages a network of interconnected devices and sensors to optimize urban transportation. By collecting real-time data on traffic conditions, environmental factors, and vehicle movements, IoT systems enable authorities to improve traffic flow, enhance safety, and promote sustainability. IoT technology also empowers drivers with timely information on congestion and alternate routes. While it holds great promise for creating smarter, more efficient cities, addressing data security, privacy concerns, and infrastructure challenges is essential for successful implementation.

Thank you!