ITAI 3377

Professor Patricia McManus

Hoang Dinh

L07 Summary Report

Introduction

This report analyzes Age of Information (AoI) and reliability in Industrial Internet of Things (IIoT) networks. It explores how different network settings affect these metrics and uses machine learning to study and predict network behavior. The analysis is based on a research-inspired dataset and includes data exploration, visualization, model building, and interpretation. By studying the balance between data freshness and reliable delivery, the report helps guide decisions to improve IIoT network performance.

Conceptual Understanding

Before beginning this assignment, this paper needs to be read first to understand the main concept:

Farag, H., Ali, S. M., & Stefanović, Č. (2023). On the Analysis of AoI-Reliability Tradeoff in Heterogeneous IIoT Networks. arXiv preprint arXiv:2311.13336.

This foundational paper introduces the trade-off between Age of Information (AoI) and reliability (often measured via Packet Loss Probability, PLP) in IIoT networks. The concepts outlined in the paper are critical for understanding how different types of traffic interact in a shared industrial network environment.

Age of Information (AoI) measures how fresh the data is that a system receives. It tracks the time since the last update was created. Unlike common metrics like delay or throughput, AoI directly shows how up to date the information is at the receiver. This is especially important in IIoT systems, where timely sensor updates are crucial for accurate decisions and smooth operations.

Real-world example: in a chemical plant, temperature sensors send data to a control system. If the data is outdated because of delays or loss, the system might make wrong adjustments, leading to unsafe conditions. A low AoI ensures the control system is always working with the latest information.

IIoT networks often handle two types of traffic:

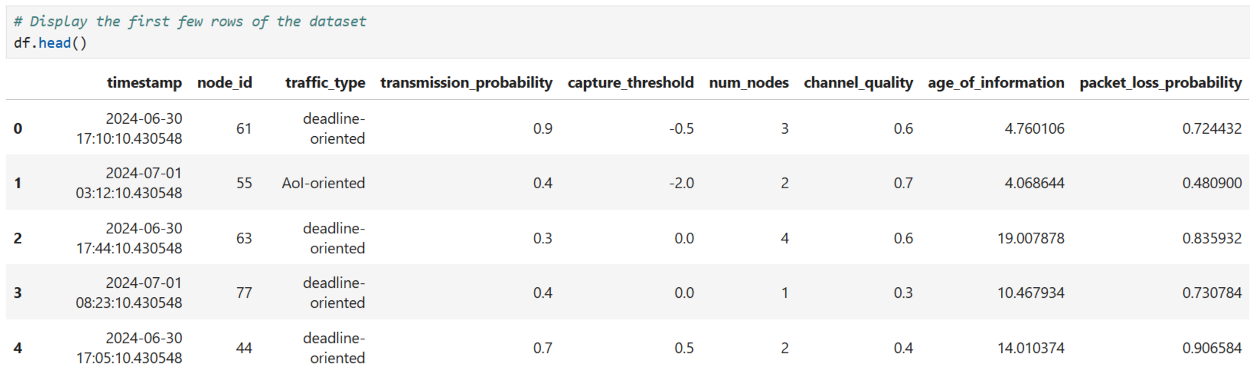
- **AoI-Oriented Traffic**: Generated periodically to keep the system updated with fresh data. It prioritizes regular and timely updates over guaranteed delivery.
- *Example*: A vibration sensor on a motor sends readings every second to monitor equipment's health. Occasional packet losses are acceptable as long as the updates are generally frequent and current.
- **Deadline-Oriented Traffic**: Triggered by specific events and must be delivered within a strict deadline to be useful. Reliability is prioritized.
- *Example*: A gas leak sensor sends an emergency alert. This packet must arrive within a second; otherwise, safety mechanisms might not be activated in time.

Understanding the difference between these traffic types is essential to analyze how AoI and PLP interact in shared network environments.

Data Exploration and Visualization

The dataset "iiot_network_data.csv" was loaded using pandas and some of its properties were explored.

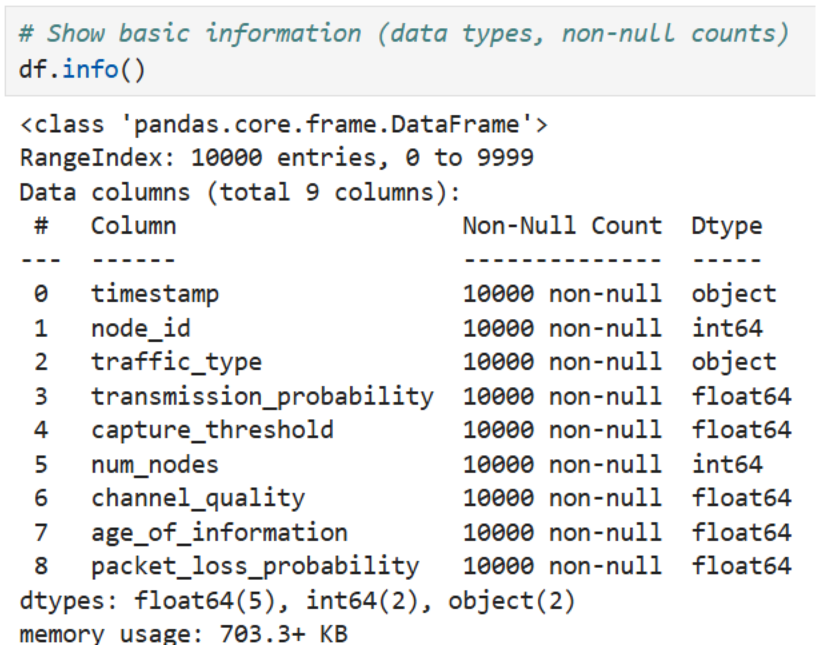
- Display the first few rows of the dataset



```
# Display the first few rows of the dataset
df.head()
```

	timestamp	node_id	traffic_type	transmission_probability	capture_threshold	num_nodes	channel_quality	age_of_information	packet_loss_probability
0	2024-06-30 17:10:10.430548	61	deadline-oriented	0.9	-0.5	3	0.6	4.760106	0.724432
1	2024-07-01 03:12:10.430548	55	AoI-oriented	0.4	-2.0	2	0.7	4.068644	0.480900
2	2024-06-30 17:44:10.430548	63	deadline-oriented	0.3	0.0	4	0.6	19.007878	0.835932
3	2024-07-01 08:23:10.430548	77	deadline-oriented	0.4	0.0	1	0.3	10.467934	0.730784
4	2024-06-30 17:05:10.430548	44	deadline-oriented	0.7	0.5	2	0.4	14.010374	0.906584

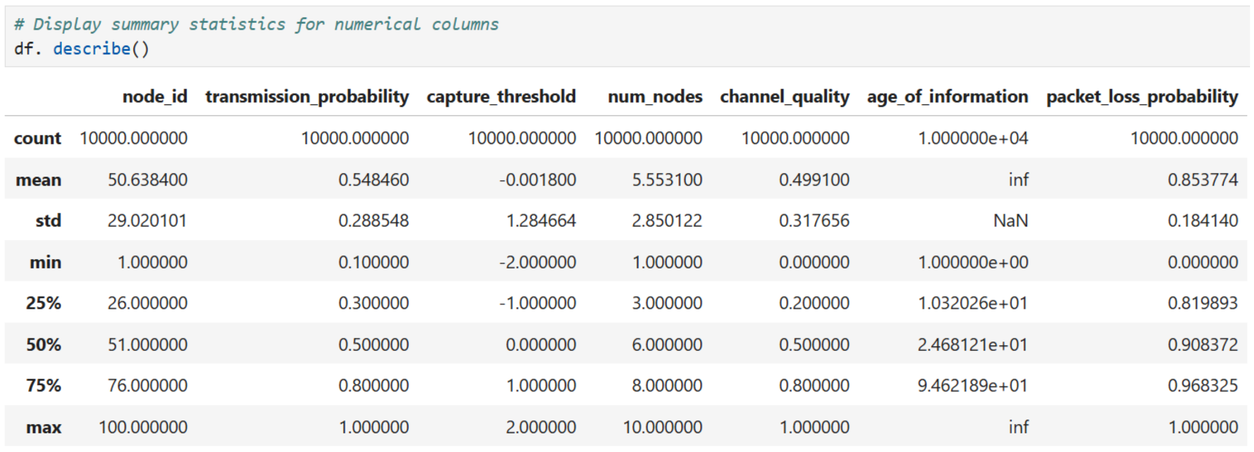
- Show basic information about the dataset (data types, non-null counts)



```python
# Show basic information (data types, non-null counts)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   timestamp                 10000 non-null  object 
 1   node_id                   10000 non-null  int64  
 2   traffic_type              10000 non-null  object 
 3   transmission_probability  10000 non-null  float64
 4   capture_threshold         10000 non-null  float64
 5   num_nodes                 10000 non-null  int64  
 6   channel_quality           10000 non-null  float64
 7   age_of_information        10000 non-null  float64
 8   packet_loss_probability   10000 non-null  float64
dtypes: float64(5), int64(2), object(2)
memory usage: 703.3+ KB
```

- Display summary statistics of the numerical columns



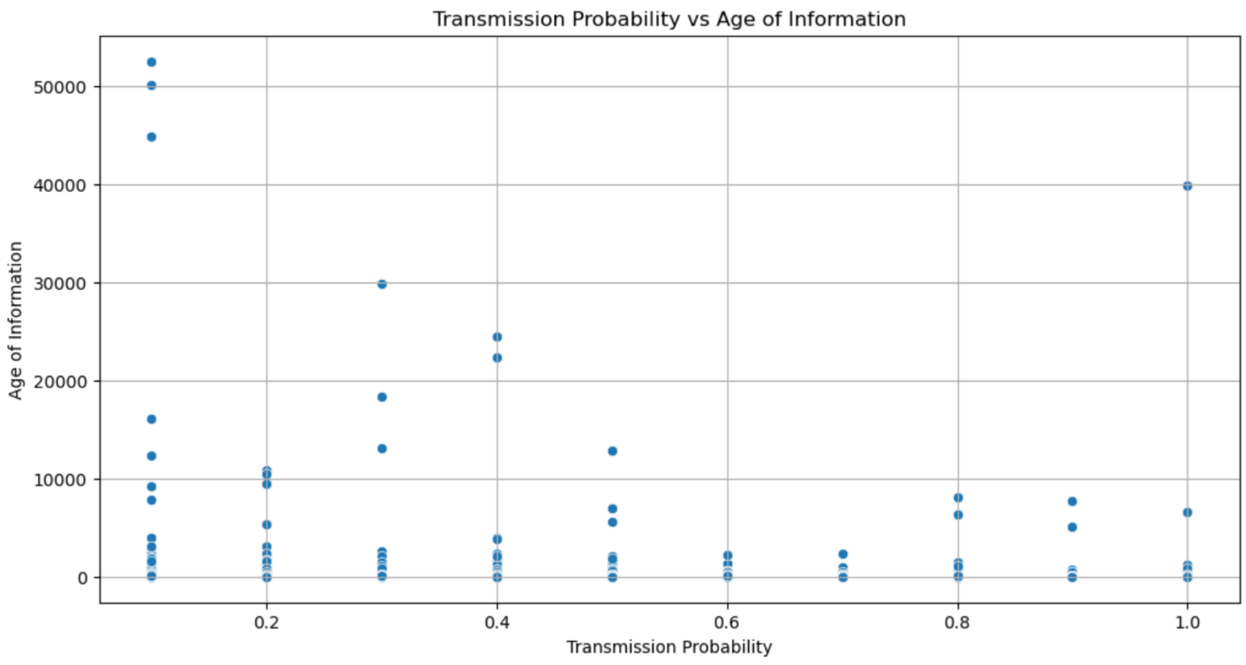
```python
# Display summary statistics for numerical columns
df. describe()
```

	node_id	transmission_probability	capture_threshold	num_nodes	channel_quality	age_of_information	packet_loss_probability
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	1.000000e+04	10000.000000
mean	50.638400	0.548460	-0.001800	5.553100	0.499100	inf	0.853774
std	29.020101	0.288548	1.284664	2.850122	0.317656	NaN	0.184140
min	1.000000	0.100000	-2.000000	1.000000	0.000000	1.000000e+00	0.000000
25%	26.000000	0.300000	-1.000000	3.000000	0.200000	1.032026e+01	0.819893
50%	51.000000	0.500000	0.000000	6.000000	0.500000	2.468121e+01	0.908372
75%	76.000000	0.800000	1.000000	8.000000	0.800000	9.462189e+01	0.968325
max	100.000000	1.000000	2.000000	10.000000	1.000000	inf	1.000000

Next, some visualizations were create for better observation:

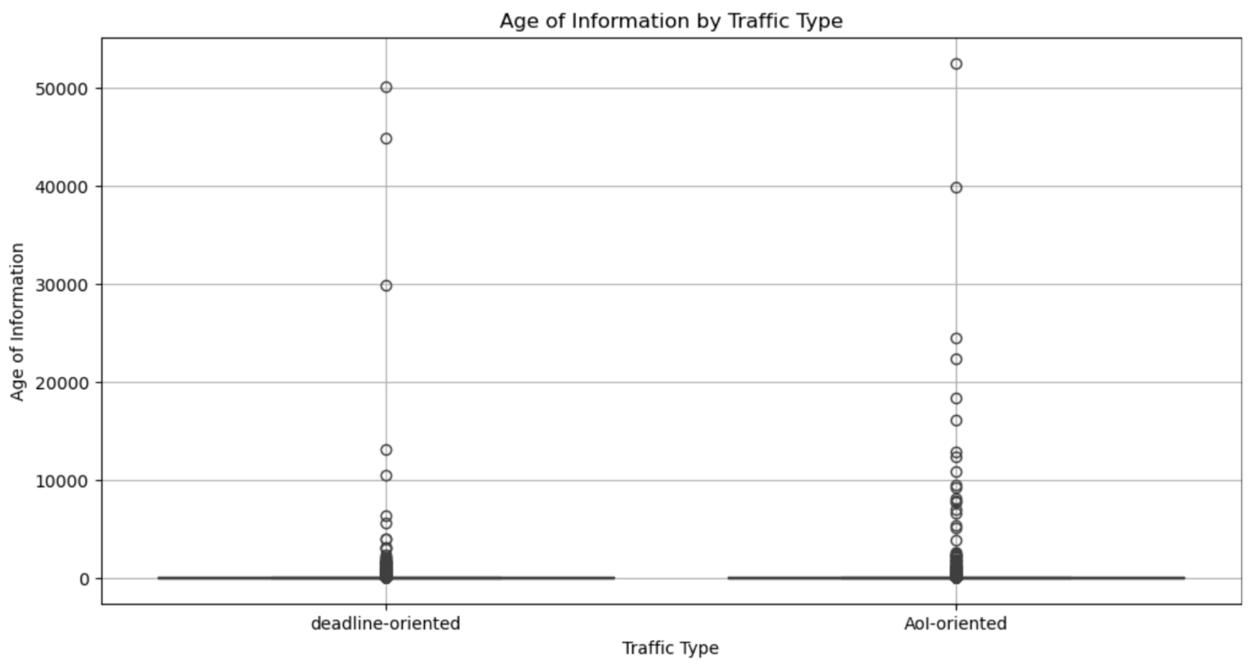
- Scatter plot of transmission_probability vs age_of_information

```
# Create visualizations
plt.figure(figsize=(12, 6))
# Add your code here to create a scatter plot of transmission_probability vs age_of_information
sns.scatterplot(data=df, x='transmission_probability', y='age_of_information')
plt.title('Transmission Probability vs Age of Information')
plt.xlabel('Transmission Probability')
plt.ylabel('Age of Information')
plt.grid(True)
plt.show()
```



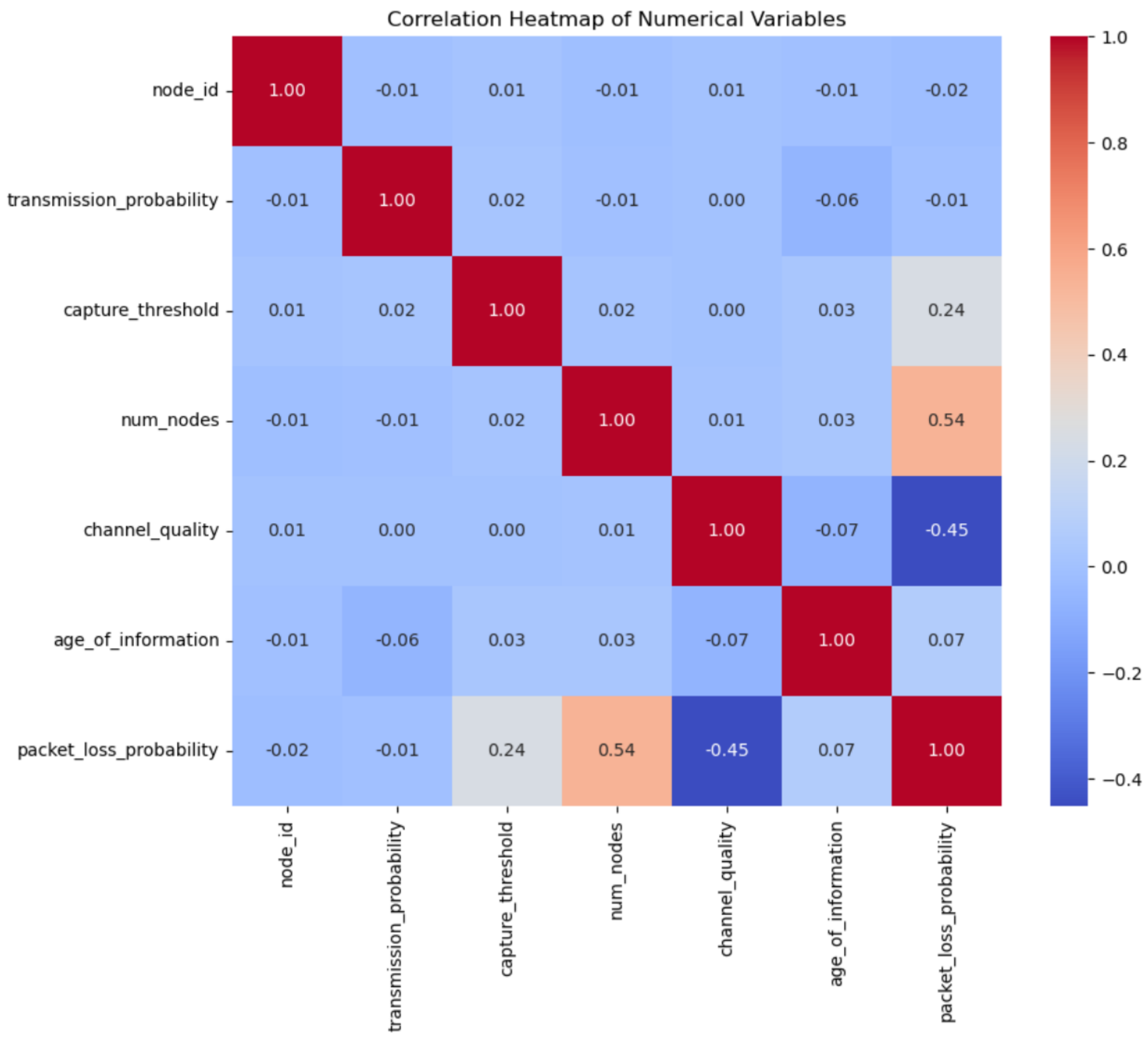
- Box plot of age_of_information grouped by traffic_type

```
plt.figure(figsize=(12, 6))
# Add your code here to create another relevant visualization
sns.boxplot(data=df, x='traffic_type', y='age_of_information')
plt.title('Age of Information by Traffic Type')
plt.xlabel('Traffic Type')
plt.ylabel('Age of Information')
plt.grid(True)
plt.show()
```



- Heatmap of correlations between numerical variables

```
# Heatmap of correlations between numerical variables
plt.figure(figsize=(12, 6))
# Compute correlation matrix
corr_matrix = df.corr(numeric_only=True) # calculates pairwise correlations between all numerical columns

# Create the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Heatmap of Numerical Variables')
plt.show()
```



Observations:

The scatter plot showed a small drop in AoI as transmission probability increased, with a weak negative correlation of -0.062. This means higher transmission probability might slightly lower AoI, but the effect is minor due to factors like network competition and packet collisions.

The box plot demonstrated clear differences between traffic types: deadline-oriented traffic had lower and more consistent AoI, while AoI-oriented traffic was more variable and generally higher. This aligns with their design—deadline-oriented packets are resent until they succeed or expire, while AoI-oriented packets are sent once, focusing on freshness rather than guaranteed delivery.

The heatmap revealed that other numerical factors have a very weak link to AoI, with packet loss probability showing the strongest connection at just 0.0685. This suggests AoI is influenced by multiple factors working together, rather than one main cause.

Overall, the results highlight that managing AoI and PLP together through effective access control, scheduling, and traffic-aware strategies is vital. Achieving good IIoT network performance requires balancing data freshness and reliability with adaptive and integrated solutions.

Machine Learning Model Development

To build a predictive model for Age of Information (AoI), several preprocessing steps were conducted. First, four relevant features were selected: transmission_probability, capture_threshold, num_nodes, and channel_quality. These features were chosen based on domain knowledge and their potential influence on AoI. The target variable was set as age_of_information.

```
# Prepare the data
X = df[['transmission_probability', 'capture_threshold', 'num_nodes', 'channel_quality']]
y_aoi = df['age_of_information']
```

A crucial additional step was data cleaning. This involved removing any rows that contained missing (NaN) or infinite values, which could otherwise cause errors during model training or lead to inaccurate predictions. Cleaning the data improves model stability and ensures the quality of input, which is especially important in real-world IIoT scenarios where sensor errors or data transmission issues may introduce anomalies into the dataset.

```
# Combine X and y for safe filtering
df_clean = df[['transmission_probability', 'capture_threshold', 'num_nodes', 'channel_quality', 'age_of_information']].copy()

# Replace inf with NaN, then drop rows with NaN
df_clean.replace([np.inf, -np.inf], np.nan, inplace=True)
df_clean.dropna(inplace=True)

# Redefine features and target
X = df_clean[['transmission_probability', 'capture_threshold', 'num_nodes', 'channel_quality']]
y_aoi = df_clean['age_of_information']
```

Next, the dataset was split into training and testing sets using an 80/20 ratio to ensure reliable evaluation of model performance. Before training, feature scaling was applied using

StandardScaler to normalize the values. This ensures that all input features contribute equally during model training and prevents features with larger scales from dominating the learning process.

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y_aoi, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
# Add your code here to fit the model
rf_model.fit(X_train_scaled, y_train)
```

To evaluate the predictive capability of the selected features, a RandomForestRegressor was initialized with 100 estimators and a fixed random state to ensure reproducibility. The model was then trained on the scaled training dataset.

```
# Make predictions
# Add your code here to make predictions on the test set
y_pred = rf_model.predict(X_test_scaled)
```

After training, predictions were made on the test set, and the model's performance was assessed using two common evaluation metrics: Mean Squared Error (MSE) and R-squared (R^2) score.

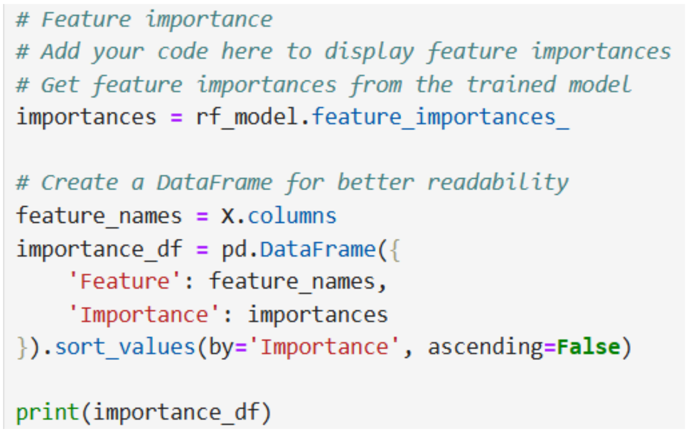
```
# Evaluate the model
# Add your code here to calculate MSE and R2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

The model yielded an MSE of 3,423,000.63 and an R^2 score of -0.55.

```
Mean Squared Error (MSE): 3423000.63
R² Score: -0.55
```

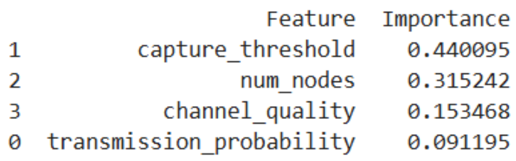
These results show that the model didn't perform well. An R^2 score of -0.55 means the model's predictions were less accurate than simply using the average AoI for all cases. The high MSE also indicates large differences between the predicted and actual AoI values. This suggests that while the chosen features are relevant, they might not be enough to model AoI accurately. It's also possible that the data relationships are too complex for a random forest model to handle without more tuning or feature improvements.

To understand which features had the most influence on the model's predictions, feature importance scores were extracted from the trained RandomForestRegressor. These scores reflect how much each feature contributed to reducing prediction error.



```
# Feature importance
# Add your code here to display feature importances
# Get feature importances from the trained model
importances = rf_model.feature_importances_

# Create a DataFrame for better readability
feature_names = X.columns
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(importance_df)
```

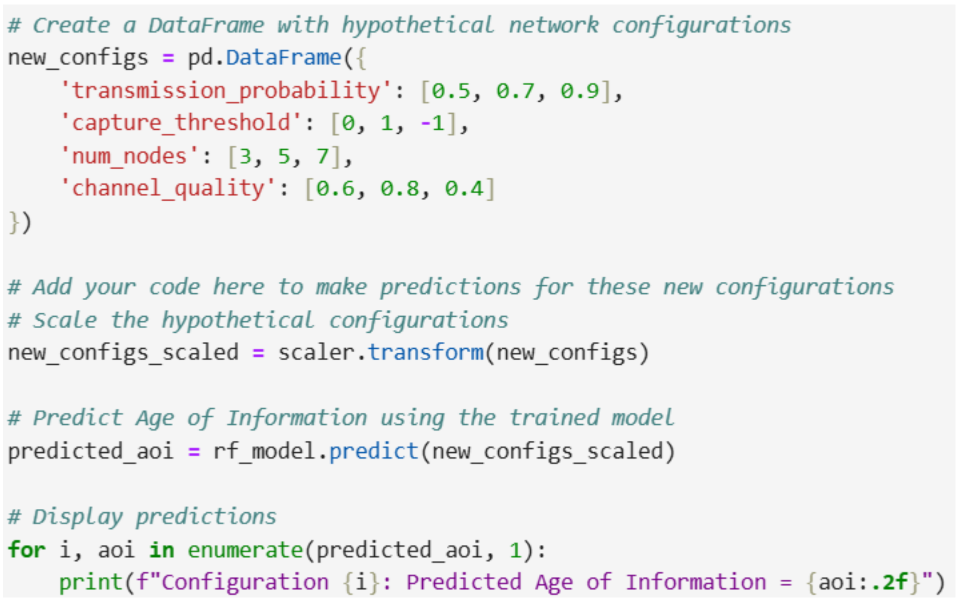
The analysis showed that capture_threshold was the most significant predictor of AoI, accounting for 44% of the total importance. This suggests that the physical-layer sensitivity to signal quality plays a major role in how fresh the information remains at the receiver. The second most important feature was num_nodes at 31.5%, indicating that network congestion and contention have a strong impact on AoI. channel_quality followed with 15.3%, showing moderate influence, while transmission_probability had the least impact at 9.1%.



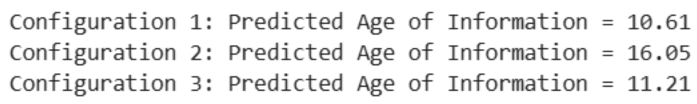
```
                  Feature  Importance
1       capture_threshold    0.440095
2               num_nodes    0.315242
3         channel_quality    0.153468
0  transmission_probability    0.091195
```

These results support the earlier visualizations: AoI is influenced by several network parameters working together, rather than just one. Recognizing how these factors interact can help focus optimization efforts in practical IIoT scenarios.

To generate predictions, a new DataFrame was created containing three hypothetical network configurations, each defined by values for transmission_probability, capture_threshold, num_nodes, and channel_quality. These configurations were scaled using the same StandardScaler that was previously fitted to the training dataset. This step ensured that the input features were normalized in the same way as during model training. After scaling, the trained RandomForestRegressor model was applied to these inputs to predict the Age of Information (AoI).



```python
# Create a DataFrame with hypothetical network configurations
new_configs = pd.DataFrame({
    'transmission_probability': [0.5, 0.7, 0.9],
    'capture_threshold': [0, 1, -1],
    'num_nodes': [3, 5, 7],
    'channel_quality': [0.6, 0.8, 0.4]
})

# Add your code here to make predictions for these new configurations
# Scale the hypothetical configurations
new_configs_scaled = scaler.transform(new_configs)

# Predict Age of Information using the trained model
predicted_aoi = rf_model.predict(new_configs_scaled)

# Display predictions
for i, aoi in enumerate(predicted_aoi, 1):
    print(f"Configuration {i}: Predicted Age of Information = {aoi:.2f}")
```



```
Configuration 1: Predicted Age of Information = 10.61
Configuration 2: Predicted Age of Information = 16.05
Configuration 3: Predicted Age of Information = 11.21
```

These predictions match the insights from both the visualizations and feature importance analysis. Configuration 2 had the highest AoI due to its high capture threshold and larger number of nodes, which led to fewer successful packets and more network congestion. Configuration 3, despite having worse channel quality, achieved a lower AoI because it had a better capture threshold and moderate traffic levels. Configuration 1 had the lowest AoI, showing a well-balanced network with fewer nodes and a neutral capture threshold. These results highlight how network parameters work together in complex ways to affect AoI. Lower capture thresholds and moderate traffic loads generally help keep data fresher.

Analysis and Insights

The balance between Age of Information (AoI) and Packet Loss Probability (PLP) depends on several interacting factors. The capture threshold, which sets the required signal strength for successful packet reception, is one key factor. A higher threshold improves reliability by reducing errors but makes it harder to receive packets, increasing AoI. The number of nodes also affects performance—more nodes lead to more competition for network access, causing delays, higher AoI, and more packet collisions, which raise PLP.

Transmission probability has a mixed effect. Increasing it can lower AoI by sending packets more often but may increase PLP due to congestion. Channel quality also matters—better conditions reduce PLP and help lower AoI by allowing smoother communication.

In summary, no single factor controls everything. Managing AoI and PLP requires balancing multiple elements, like adjusting transmission probabilities, scheduling, and capture thresholds, to keep data fresh and reliable in IIoT networks.

Some strategies for optimizing network performance to balance data freshness and reliability:

- Adjust the transmission probability based on the current state of the network. For instance, lower the access rate during congestion to reduce collisions, or raise it when the network isn't busy to keep the data fresher.
- Adjust the physical-layer settings based on the link quality. When the channel is clear, lowering the capture threshold can help reduce packet loss and improve AoI. In noisy conditions, raising the threshold can avoid incorrect decoding.
- Use priority-based scheduling methods, like assigning specific time slots or using round-robin with weighted priorities, to make sure deadline-sensitive traffic is delivered reliably. At the same time, ensure AoI-sensitive traffic gets frequent updates without being blocked or delayed.
- Create efficient queueing strategies that restrict how many deadline-oriented packets are resent to prevent network congestion. At the same time, allow outdated AoI-oriented packets to be dropped, freeing up bandwidth for newer, fresher data.

Potential real-world applications of your insights in an IIoT context:

- In industries working in remote locations (like pipelines or power grids), network bandwidth is often limited. By optimizing AoI and PLP, it's possible to monitor infrastructure status on time and make sure critical alerts (such as voltage spikes or pressure drops) are delivered reliably, even over poor or unstable connections.
- Automated forklifts and delivery robots in warehouses rely on up-to-date data, like location tracking and obstacle detection, to move safely. At the same time, critical control signals need to be sent reliably to prevent collisions. Using flexible access and scheduling methods helps keep operations smooth and safe.
- In risky places like chemical plants or oil refineries, sensors need to send alerts about gas leaks or pressure spikes with high dependability. At the same time, data like humidity or temperature must be kept up-to-date to prevent dangerous changes. Managing AoI and PLP together ensures both kinds of data work properly.

Conclusions

The project shows that AoI and PLP are influenced by the complex interplay of network settings. While machine learning offers helpful insights, achieving the best balance between data freshness and reliability in IIoT systems requires system-level planning and strategies.