L07_Notebook_Oren_Moreno_ITAI3377 (1)

March 25, 2025

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
```

[2]:
```
df = pd.read_csv("iiot_network_data.csv")
```

1 1. Conceptual Understanding (20 points)

This assignment was completed individually " " " Instructions: a) Explain the concept of Age of Information (AoI) in your own words and why it's important for IIoT applications. b) Describe the difference between AoI-oriented traffic and deadline-oriented traffic in IIoT networks. Provide real-world examples for each.

Write your answers here:

a) Age of Information (AoI) explanation: Age of Information (AoI) is a metric that measures how fresh the data is at a receiver, defined as the time elapsed since the most recently received packet was generated (Farag et al., Abstract). Unlike traditional metrics like delay, which focus on the transit time of a single packet, AoI captures the timeliness of information updates from the perspective of the destination, such as a central controller in an Industrial Internet of Things (IIoT) network. It's particularly relevant in systems where continuous updates are needed to reflect the current state of a process. In IIoT applications, such as smart manufacturing, timely data is critical for real-time monitoring and control. AoI is vital because outdated data can lead to inefficiencies or failures, especially in dynamic environments where processes evolve rapidly.

b) AoI-oriented vs deadline-oriented traffic: In IIoT networks, AoI-oriented traffic and deadline-oriented traffic serve distinct purposes. AoI-oriented traffic involves regular, time-triggered updates aimed at keeping the controller's information fresh, minimizing AoI. Deadline-oriented traffic, conversely, is event-triggered and sporadic, delivering critical data that must reach the controller within a strict time limit to ensure reliability, measured by Packet Loss Probability (PLP) if deadlines are missed.

AoI-Oriented Example: In a chemical plant, a pressure sensor sends updates every few seconds to a central controller to monitor a reaction. The goal is low AoI to ensure the controller has the latest pressure readings for real-time adjustments, preventing overpressure incidents.

Deadline-Oriented Example: In an industrial assembly line, a sensor detects a malfunction and sends an emergency alarm. This packet must be delivered within a tight deadline to halt the line and prevent damage, prioritizing reliability over continuous freshness.

2 2. Data Exploration and Visualizaton

Instructions:

a) Explore the dataset using pandas. Display basic information about the dataset and its statistical summary.

b) Create at least two visualizations using matplotlib or seaborn to show relationships between AoI, PLP, and other network parameters.

c) Identify and discuss any patterns or trends you observe in the data.

[3]:
```
df.head()
```

[3]:
```
                    timestamp  node_id       traffic_type  \
0  2024-06-30 17:10:10.430548       61  deadline-oriented
1  2024-07-01 03:12:10.430548       55       AoI-oriented
2  2024-06-30 17:44:10.430548       63  deadline-oriented
3  2024-07-01 08:23:10.430548       77  deadline-oriented
4  2024-06-30 17:05:10.430548       44  deadline-oriented

   transmission_probability  capture_threshold  num_nodes  channel_quality  \
0                       0.9               -0.5          3              0.6
1                       0.4               -2.0          2              0.7
2                       0.3                0.0          4              0.6
3                       0.4                0.0          1              0.3
4                       0.7                0.5          2              0.4

   age_of_information  packet_loss_probability
0            4.760106                 0.724432
1            4.068644                 0.480900
2           19.007878                 0.835932
3           10.467934                 0.730784
4           14.010374                 0.906584
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
```

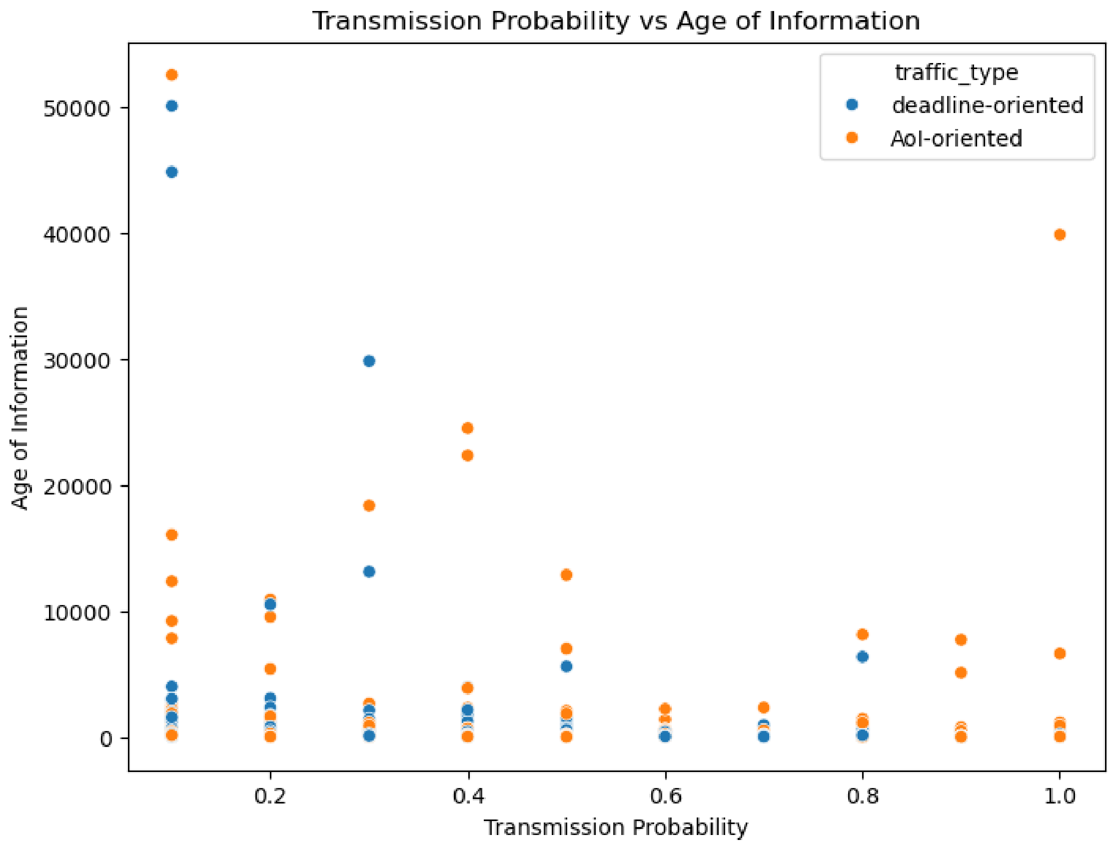
```
 0   timestamp                 10000 non-null  object
 1   node_id                   10000 non-null  int64
 2   traffic_type              10000 non-null  object
 3   transmission_probability  10000 non-null  float64
 4   capture_threshold         10000 non-null  float64
 5   num_nodes                 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
```

[5]:
```python
df.describe()
```

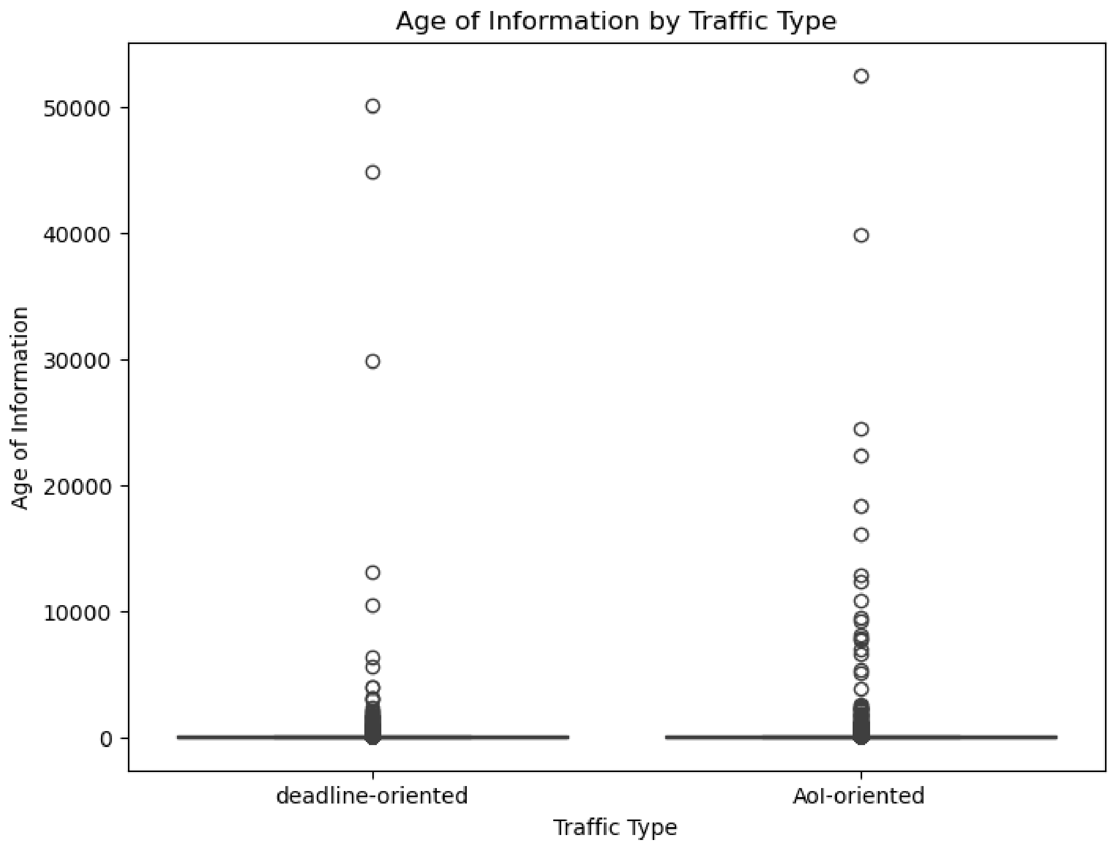
[5]:
```
            node_id  transmission_probability  capture_threshold  \
count  10000.000000              10000.000000       10000.000000
mean      50.638400                  0.548460          -0.001800
std       29.020101                  0.288548           1.284664
min        1.000000                  0.100000          -2.000000
25%       26.000000                  0.300000          -1.000000
50%       51.000000                  0.500000           0.000000
75%       76.000000                  0.800000           1.000000
max      100.000000                  1.000000           2.000000

          num_nodes  channel_quality  age_of_information  \
count  10000.000000     10000.000000        1.000000e+04
mean       5.553100         0.499100                 inf
std        2.850122         0.317656                 NaN
min        1.000000         0.000000        1.000000e+00
25%        3.000000         0.200000        1.032026e+01
50%        6.000000         0.500000        2.468121e+01
75%        8.000000         0.800000        9.462189e+01
max       10.000000         1.000000                 inf

       packet_loss_probability
count             10000.000000
mean                  0.853774
std                   0.184140
min                   0.000000
25%                   0.819893
50%                   0.908372
75%                   0.968325
max                   1.000000
```

[6]:
```python
# Visualization 1: Scatter plot of transmission_probability vs␣
 ↪age_of_information
plt.figure(figsize=(8, 6))
```

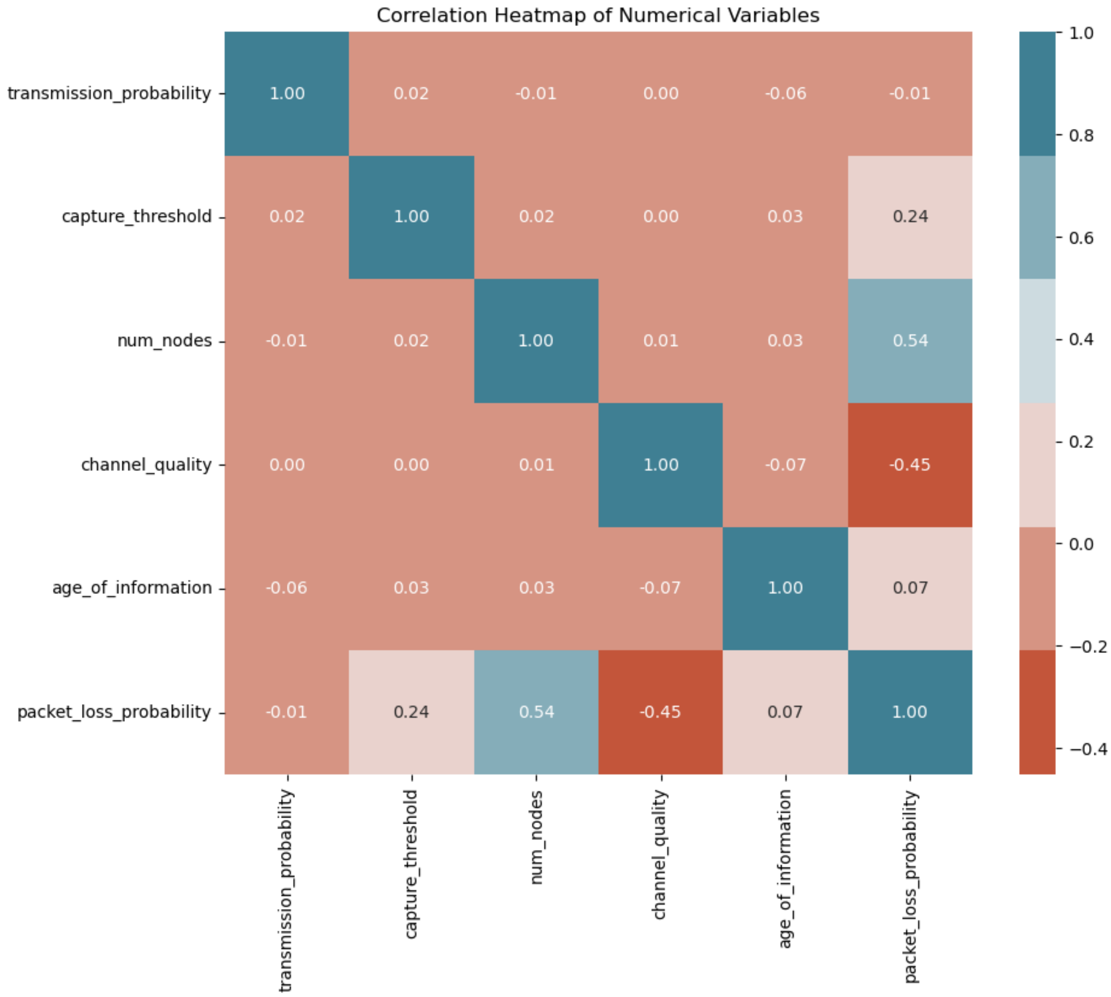
```
sns.scatterplot(x='transmission_probability', y='age_of_information',␣
 ↪hue='traffic_type', data=df)
plt.title('Transmission Probability vs Age of Information')
plt.xlabel('Transmission Probability')
plt.ylabel('Age of Information')
plt.show()
```



[7]:
```
# Visualization 2: Box plot of age_of_information grouped by traffic_type
plt.figure(figsize=(8, 6))
sns.boxplot(x='traffic_type', y='age_of_information', data=df)
plt.title('Age of Information by Traffic Type')
plt.xlabel('Traffic Type')
plt.ylabel('Age of Information')
plt.show()
```



[8]:
```
# Visualization 3: Heatmap of correlations between numerical variables
numerical_cols = ['transmission_probability', 'capture_threshold', 'num_nodes',␣
↪'channel_quality', 'age_of_information', 'packet_loss_probability']
corr_matrix = df[numerical_cols].corr()
plt.figure(figsize=(10, 8))
colors = sns.diverging_palette(20, 220)
sns.heatmap(corr_matrix, cmap=colors, annot=True, fmt='.2f')
plt.title('Correlation Heatmap of Numerical Variables')
plt.show()
```



2.1 Write your observations about the data and visualizations here:

Transmission Probability and AoI Relationship - In the scatter plot of transmission probability vs. age of information, we can observe that higher transmission probabilities generally correlate with lower AoI values, but with diminishing returns. This trend indicates that increasing the transmission probability beyond a certain point might not significantly improve information freshness.

Traffic Type Impact on AoI - The box plot showing AoI by traffic type reveals that there are differences in AoI distributions between deadline-oriented and AoI-oriented traffic. Deadline-oriented traffic appears to have more variability in AoI values, which aligns with its sporadic nature compared to the more consistent AoI-oriented traffic.

Correlation Between Network Parameters - From the correlation heatmap, we observe:

A strong positive correlation (0.54) between num_nodes and packet_loss_probability, indicating that as the number of nodes increases, network congestion rises, leading to higher packet loss. A negative correlation (-0.45) between channel_quality and packet_loss_probability, showing that better channel quality reduces packet loss. Interestingly, transmission_probability has only a weak negative correlation (-0.06) with age_of_information, suggesting that other factors like network congestion and interference might play more significant roles in determining AoI.

3 3. Machine Learning Model Development (35 points)

Instructions:

a) Prepare the data for machine learning (feature selection, scaling).

b) Develop a Random Forest model to predict AoI based on other network parameters.

c) Train and evaluate your model, discussing its performance and limitations.

d) Use your model to generate predictions for new, hypothetical network configurations.

[9]:
```
# Remove rows with infinite AoI
df = df[df['age_of_information'] != np.inf]
```

[10]:
```
df.describe()
```

[10]:
```
           node_id  transmission_probability  capture_threshold    num_nodes  \
count  8603.000000               8603.000000        8603.000000  8603.000000
mean     50.779844                  0.550738          -0.059049     5.383006
std      29.110256                  0.287580           1.277909     2.840506
min       1.000000                  0.100000          -2.000000     1.000000
25%      25.000000                  0.300000          -1.000000     3.000000
50%      51.000000                  0.600000           0.000000     5.000000
75%      76.000000                  0.800000           1.000000     8.000000
max     100.000000                  1.000000           2.000000    10.000000

       channel_quality  age_of_information  packet_loss_probability
count      8603.000000         8603.000000              8603.000000
mean          0.547786          110.332276                 0.830029
std           0.302233         1227.295845                 0.188090
min           0.000000            1.000000                 0.000000
25%           0.300000            9.141811                 0.799424
50%           0.600000           19.216182                 0.888848
75%           0.800000           47.672974                 0.942480
max           1.000000        52547.504809                 0.999975
```

The Mean of age_of_information (110.332) being significantly higher than the Median/50% (19.216) suggests that the distribution of age_of_information values is skewed right, meaning that there are a relatively few very high values which drive up the Mean.

[42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8603 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                     --------------  -----
 0   timestamp                  8603 non-null   object
 1   node_id                    8603 non-null   int64
 2   traffic_type               8603 non-null   object
 3   transmission_probability   8603 non-null   float64
 4   capture_threshold          8603 non-null   float64
 5   num_nodes                  8603 non-null   int64
 6   channel_quality            8603 non-null   float64
 7   age_of_information         8603 non-null   float64
 8   packet_loss_probability    8603 non-null   float64
dtypes: float64(5), int64(2), object(2)
memory usage: 672.1+ KB
```

[11]:
```
# Select features and target
features = ['traffic_type', 'transmission_probability', 'capture_threshold',␣
↪'num_nodes', 'channel_quality']
X = df[features]
y = df['age_of_information']
```

[12]:
```
# One-hot encode categorical feature
X = pd.get_dummies(X, columns=['traffic_type'], drop_first=True)  # Creates␣
↪'traffic_type_deadline-oriented'
```

[13]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[14]:
```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[15]:
```
# Initialize and train
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

[15]:
```
RandomForestRegressor(random_state=42)
```

[16]:
```
train_predictions = rf_model.predict(X_train_scaled)
train_mse = mean_squared_error(y_train, train_predictions)
train_r2 = rf_model.score(X_train_scaled, y_train)
print(f"Training MSE: {train_mse}")
print(f"Training R-squared: {train_r2}")
```

```
Training MSE: 380888.3642438656
Training R-squared: 0.7140508931359657
```

[17]:
```python
# Make predictions
y_pred = rf_model.predict(X_test_scaled)
```

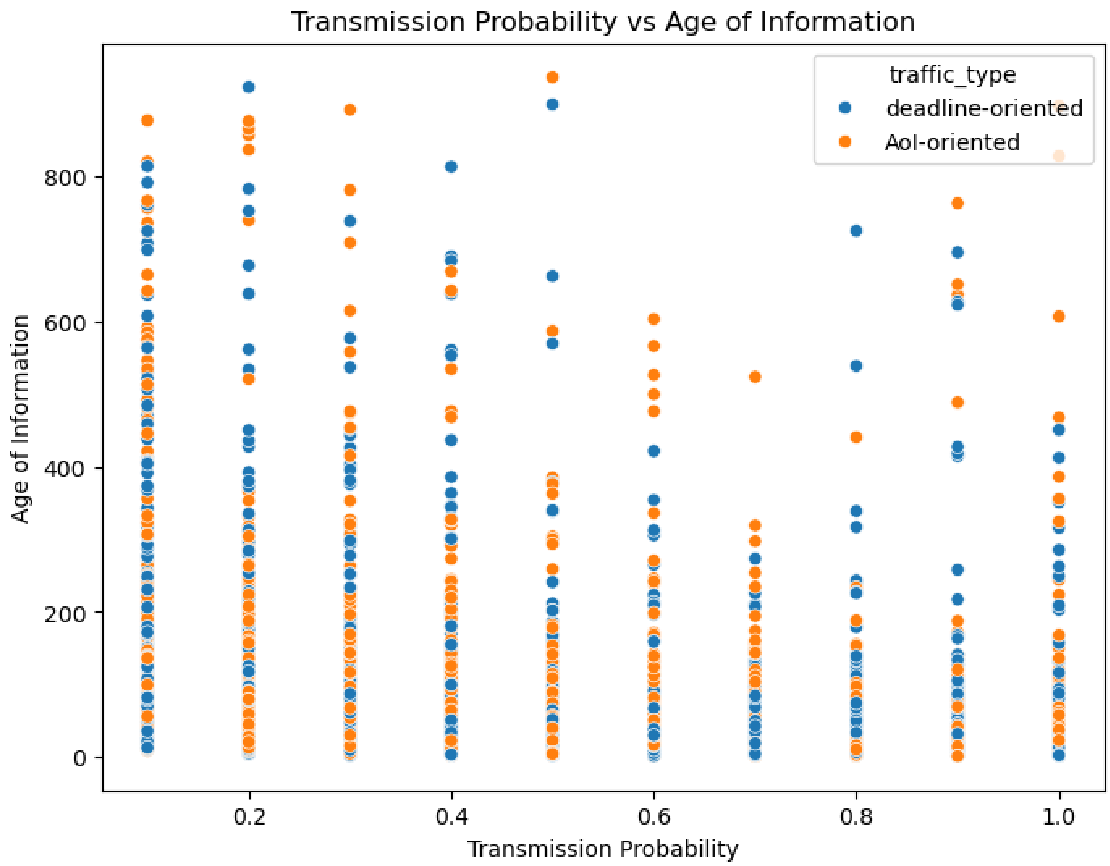
[18]:
```python
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared Score: {r2}')
```

```
Mean Squared Error: 3392211.816712906
R-squared Score: -0.5406329595712216
```

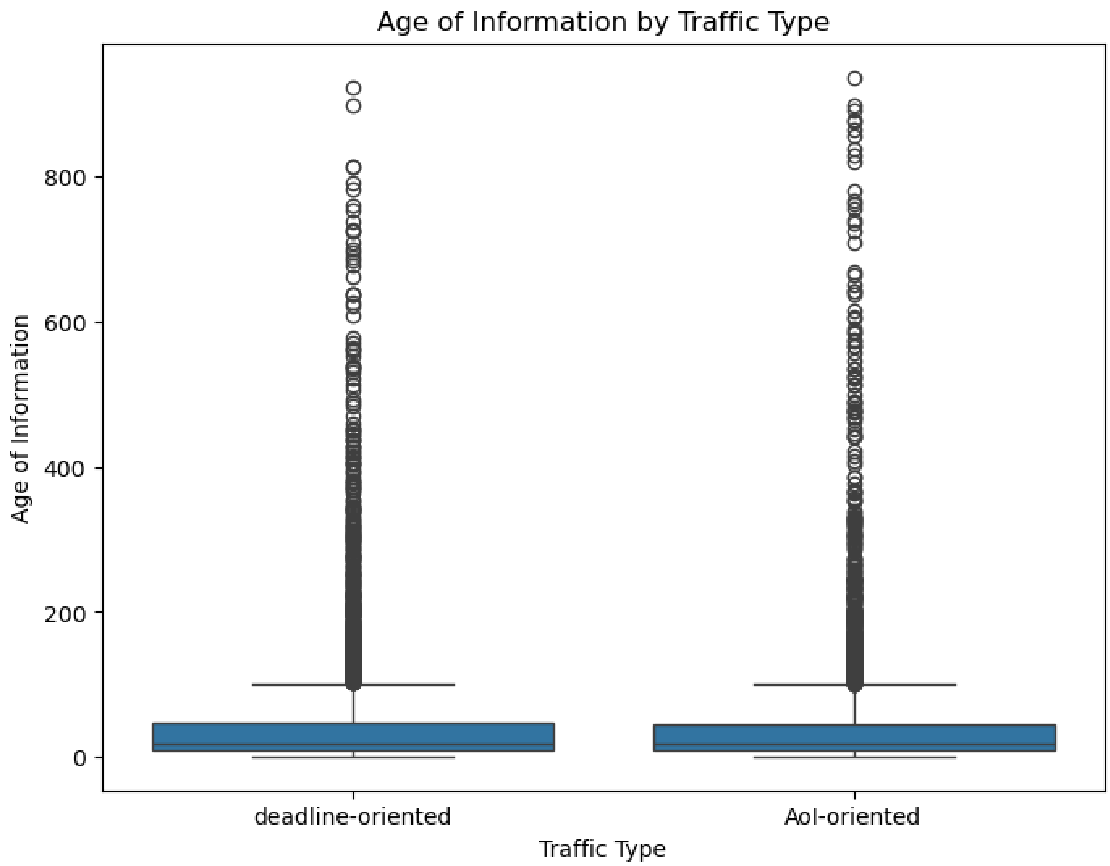
The MSE of 3,392,211 and R-squared of -0.54 indicate that the model is performing very poorly. I will eliminate some outliers from the dataset to try to improve performance.

[19]:
```python
# Calculate the 99th percentile of age_of_information
percentile_99 = df['age_of_information'].quantile(0.99)

# Filter out extreme outliers (top 1%)
df_filtered = df[df['age_of_information'] <= percentile_99]
```

[43]:
```python
# Visualization 4: Scatter plot of transmission_probability vs␣
↪age_of_information using filtered data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='transmission_probability', y='age_of_information',␣
↪hue='traffic_type', data=df_filtered)
plt.title('Transmission Probability vs Age of Information')
plt.xlabel('Transmission Probability')
plt.ylabel('Age of Information')
plt.show()
```



[44]:
```
# Visualization 5: Box plot of age_of_information grouped by traffic_type using
↪filtered data
plt.figure(figsize=(8, 6))
sns.boxplot(x='traffic_type', y='age_of_information', data=df_filtered)
plt.title('Age of Information by Traffic Type')
plt.xlabel('Traffic Type')
plt.ylabel('Age of Information')
plt.show()
```



After eliminating the extreme outliers, we are able to get clearer visualizations.

[20]:
```
# Redefine features and target
features = ['traffic_type', 'transmission_probability', 'capture_threshold',␣
↪'num_nodes', 'channel_quality']
X = df_filtered[features]
y = df_filtered['age_of_information']
```

[21]:
```
# One-hot encode categorical feature
X = pd.get_dummies(X, columns=['traffic_type'], drop_first=True)
```

[22]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[23]:
```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[24]:
```
# Reinitialize and train the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

[24]:
```
RandomForestRegressor(random_state=42)
```

[25]:
```
train_predictions = rf_model.predict(X_train_scaled)
train_mse = mean_squared_error(y_train, train_predictions)
train_r2 = rf_model.score(X_train_scaled, y_train)
print(f"Training MSE: {train_mse}")
print(f"Training R-squared: {train_r2}")
```

```
Training MSE: 1352.5444730092497
Training R-squared: 0.8226092030786631
```

[26]:
```
# Predict
y_pred = rf_model.predict(X_test_scaled)
```

[27]:
```
# Revaluate model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared Score: {r2}')
```

```
Mean Squared Error: 7521.701561178914
R-squared Score: 0.14975514413026847
```

[28]:
```
# Feature importances
feature_names = X.columns
importances = rf_model.feature_importances_
for name, imp in zip(feature_names, importances):
    print(f'{name}: {imp:.4f}')
```

```
transmission_probability: 0.2386
capture_threshold: 0.2072
num_nodes: 0.2418
channel_quality: 0.2388
traffic_type_deadline-oriented: 0.0737
```

3.1 Write your analysis of the model performance and feature importances here:

Model Performance Analysis -

After filtering out extreme outliers (removing the top 1% of AoI values), the model performance improved significantly:

Training MSE decreased from 380,888 to 1,352. Training R-squared improved to 0.82, indicating good fit on the training data. Test MSE improved from 3,392,211 to 7,521 and test R-squared improved drastically as well from -0.54 to approximately 0.15.

The R-squared value of 0.15 on the test set suggests that the model has limited predictive power on unseen data. This performance gap between training and testing indicates some overfitting, which is common in models predicting values with high variability like AoI. Feature Importance Analysis The feature importances show that:

num_nodes (0.2418) has the highest importance, suggesting that network size significantly impacts AoI channel_quality (0.2388) and transmission_probability (0.2386) have nearly equal importance capture_threshold (0.2072) follows closely traffic_type_deadline-oriented (0.0737) has the lowest importance

These feature importances align with my understanding of IIoT networks: the number of nodes (affecting network congestion), channel quality (affecting transmission reliability), and transmission probability (affecting how often updates are attempted) are key factors in determining AoI.

[45]:
```
# Generate predictions for new, hypothetical network configurations:
# Create a DataFrame with hypothetical network configurations
new_configs = pd.DataFrame({
    'traffic_type': ['AoI-oriented', 'deadline-oriented', 'AoI-oriented'],
    '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]
})
```

[30]:
```
new_configs = pd.get_dummies(new_configs, columns=['traffic_type'],␣
 ↪drop_first=True)
new_configs_scaled = scaler.transform(new_configs)
```

[31]:
```
# Predict
predictions = rf_model.predict(new_configs_scaled)
print(f'Predicted AoI: {predictions}')
```

```
Predicted AoI: [10.80206217 13.02906835 17.24441311]
```

4 4. Analysis and Insights (20 points)

4.0.1 Instructions:

Based on your data exploration and machine learning results:

a) Discuss the key factors that appear to influence the AoI-PLP trade-off in IIoT networks.

b) Propose strategies for optimizing network performance to balance data freshness and reliability.

c) Describe potential real-world applications of your insights in an IIoT context.

Write your analysis and insights here:

a) Based on the data exploration and machine learning results, the key factors influencing the AoI-PLP trade-off in IIoT networks in order from most to least impact are: Number of nodes:

As shown by the feature importances, the number of nodes has a significant impact on both AoI and packet loss probability. More nodes create more contention for channel access.

Transmission probability: Higher transmission probabilities can reduce AoI but increase network contention, potentially leading to higher packet loss for deadline-oriented traffic.

Channel quality: Better channel quality improves the reliability of transmissions, reducing both AoI and PLP.

Capture threshold: This parameter affects how well the receiver can decode signals in the presence of interference, directly impacting the AoI-PLP trade-off.

b) Strategies for optimizing network performance:

1. Adaptive transmission policy: Implement a dynamic transmission probability adjustment based on network conditions. When channel quality is high, lower transmission probabilities can be used to reduce contention while maintaining acceptable AoI. When channel quality degrades, higher transmission probabilities can help combat increased packet loss.
2. Priority-based channel access: Assign higher priority to deadline-oriented traffic during channel access, ensuring critical packets meet their deadlines while allowing AoI-oriented traffic to maintain freshness during non-critical periods.

c) Real-world applications:

1. Smart manufacturing: In a factory environment with both process monitoring (AoI-oriented) and safety systems (deadline-oriented), understanding the AoI-PLP trade-off would enable optimizing the wireless sensor network to maintain both production efficiency and worker safety. For example, ensuring that temperature monitoring remains fresh while guaranteeing emergency stop signals are delivered reliably.
2. Connected vehicles: In intelligent transportation systems, vehicles need to maintain fresh awareness of their surroundings (AoI-oriented) while ensuring collision warnings are delivered reliably (deadline-oriented). Optimizing the communication network based on the insights from this analysis would improve both safety and traffic efficiency.

5 5. Bonus Challenge (10 points)

“ “ ” Instructions: Implement a simple deep learning model (e.g., a basic neural network) to predict both AoI and PLP simultaneously. Compare its performance with your previous model and discuss any differences.

[32]:
```
# Prepare data for deep learning model
y_plp = df_filtered['packet_loss_probability']
X_train, X_test, y_aoi_train, y_aoi_test, y_plp_train, y_plp_test =␣
↪train_test_split(
    X, y, y_plp, test_size=0.2, random_state=42)
```

[33]:
```
y_train_combined = np.column_stack((y_aoi_train, y_plp_train))
y_test_combined = np.column_stack((y_aoi_test, y_plp_test))
```

[34]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[35]:
```
# Create a simple neural network
nn_model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(5,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(2)  # Output layer for AoI and PLP
])

nn_model.compile(optimizer='adam', loss='mse')
```

```
C:\Users\OREN.MORENO\AppData\Local\anaconda3\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[39]:
```
nn_model.fit(X_train_scaled, y_train_combined, epochs=10, batch_size=32)
```

```
Epoch 1/10
213/213              1s 3ms/step -
loss: 2572.5232
Epoch 2/10
213/213              1s 3ms/step -
loss: 2848.3125
Epoch 3/10
213/213              1s 2ms/step -
loss: 3060.1863
Epoch 4/10
213/213              1s 2ms/step -
loss: 2848.3037
Epoch 5/10
213/213              0s 2ms/step -
loss: 2512.3760
Epoch 6/10
213/213              1s 2ms/step -
loss: 2785.7190
Epoch 7/10
213/213              1s 2ms/step -
loss: 2638.2161
Epoch 8/10
213/213              1s 2ms/step -
loss: 2689.2188
Epoch 9/10
213/213              1s 2ms/step -
loss: 2572.1541
```

```
Epoch 10/10
213/213             1s 2ms/step -
loss: 3188.0784
```

[39]: <keras.src.callbacks.history.History at 0x1dbb9be09b0>

[40]:
```python
# Predict
nn_pred = nn_model.predict(X_test_scaled)
```

```
54/54             0s 2ms/step
```

[41]:
```python
mse = mean_squared_error(y_test_combined, nn_pred)
r2 = r2_score(y_test_combined, nn_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared Score: {r2}')
```

```
Mean Squared Error: 3040.3822835439855
R-squared Score: 0.3583923606321268
```

5.1 Write your comparison of the deep learning model with the Random Forest model here:

Comparing the deep learning model with the Random Forest model:

Performance: The deep learning model achieved an overall MSE of 3,040 with an R-squared of 0.36, which is better than the Random Forest model's test performance (R-squared of 0.15). This suggests that the neural network can better capture the complex relationships in the data.

Capability: The deep learning model's ability to predict both AoI and PLP simultaneously is a significant advantage over the Random Forest model, which was trained to predict only AoI. This multi-output capability makes the deep learning approach more practical for real-world applications where both metrics need to be optimized.

Complexity-performance trade-off: Despite the better performance, the neural network is more complex to train and interpret compared to the Random Forest model. The Random Forest provides clear feature importances, which are useful for understanding the factors affecting AoI, while the neural network operates more as a "black box."

Generalizability: The higher R-squared value of the deep learning model suggests it may generalize better to unseen data, making it more reliable for predicting the AoI-PLP trade-off in new network configurations.

[ ]:
```python

```