1. Pay special attention to the explanations of Age of Information (AoI) and Packet Loss Probability (PLP).
2. Note how the authors model the IIoT network and the assumptions they make.
3. Understand the trade-offs discussed between AoI and reliability in IIoT networks.

Age of Information (AOI) was brought to present new way to monitor the controller and keep it up to date with better performance than delay and throughput. IIoT applications tend to be biased towards the AOI sensitive traffic flows and not the insensitive trafficflows. The point of the paper is to compare AOI and deadline-oriented traffic being represented by Packet Loss Probability (PLP) in a heterogeneous IIOT network.You can't just focus on one type of data. Emergency and monitoring data compete for limited wireless bandwidth. This paper helps system designers find a sweet spot that keeps everything running smoothly and safely.

1. Conceptual Understanding

a) In your own words, explain the concept of Age of Information (AoI). Why is it important for IIoT applications? Provide a real-world example to illustrate your explanation.

b) Describe the difference between AoI-oriented traffic and deadline-oriented traffic in IIoT networks. Provide a real-world example for each type of traffic.

I define AOI as a metric designed to understand the freshness of data. The lower the number the fresher the data is. It's really important for IIOT so it can use fast real time data for its decisions. This could be controlling machines in controlling machines, monitoring weather conditions, or cybersecurity safety. The difference between AOI and deadline oriented is the span of time. In AoI you try to keep the staleness good, but in deadline oriented you just have a strict deadline to achieve. For AOI you want things that require a constant feedback loop, where in deadline the urgency demands the movement. An example would be updating data for measuring the environment to the cloud which isn't necessarily urgent, where for deadline-oriented traffic an example could be a robot in a manufacturing plant that has to hit deadlines in milliseconds or it could misbehave.

```python
# STEP 1: Upload the file from local system (you'll get a file selector pop-up)
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>

Saving iiot_network_data.csv to iiot_network_data.csv
```

```python
# 1. Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Optional: make plots look cleaner
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)
```

```python
# 2. Load the dataset
df = pd.read_csv('/content/iiot_network_data.csv')

# 3. Display the first few rows
print("□ First 5 rows of the dataset:")
display(df.head())

# 4. Show basic info
print("\n□ Basic info about the dataset:")
df.info()

# 5. Summary statistics of numerical columns
print("\n□ Summary statistics:")
display(df.describe())
```

```
□ First 5 rows of the dataset:
```


```
{"summary":"{\n  \"name\": \"display(df\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"timestamp\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-
07-01 03:12:10.430548\",\n          \"2024-06-30 17:05:10.430548\",\n
\"2024-06-30 17:44:10.430548\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"node_id\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 12,\n        \"min\": 44,\n
\"max\": 77,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          55,\n          44,\n          63\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"traffic_type\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"AoI-
oriented\",\n          \"deadline-oriented\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"transmission_probability\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.25099800796022265,\n        \"min\": 0.3,\n        \"max\": 0.9,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          0.4,\n
0.7000000000000001\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"capture_threshold\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.9617692030835673,\n        \"min\": -
2.0,\n        \"max\": 0.5,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          -2.0,\n          0.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"num_nodes\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 1,\n        \"max\": 4,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2,\n
1\n        ],\n        \"semantic_type\": \"\",\n
```

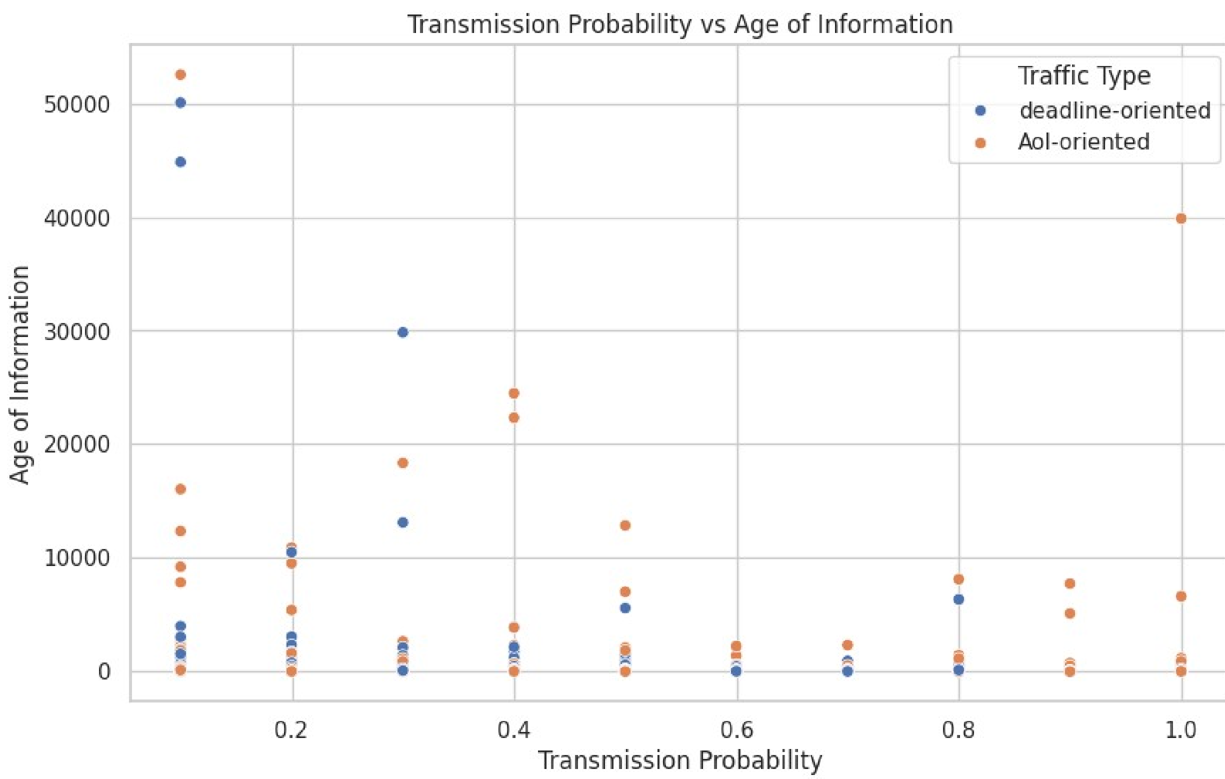
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"channel_quality\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.16431676725154987,\n   \"min\": 0.3,\n   \"max\": 0.7000000000000001,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   0.7000000000000001,\n   0.4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"age_of_information\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 6.304950510528936,\n   \"min\": 4.068644401165404,\n   \"max\": 19.00787763504463,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   4.068644401165404,\n   14.010373522931014\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"packet_loss_probability\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.16149478226311534,\n   \"min\": 0.4808996154955939,\n   \"max\": 0.9065843639866464,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.4808996154955939,\n   0.9065843639866464\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
 Basic info about the dataset:
<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

 Summary statistics:
```

{"repr_error":"Out of range float values are not JSON compliant: inf","type":"dataframe"}

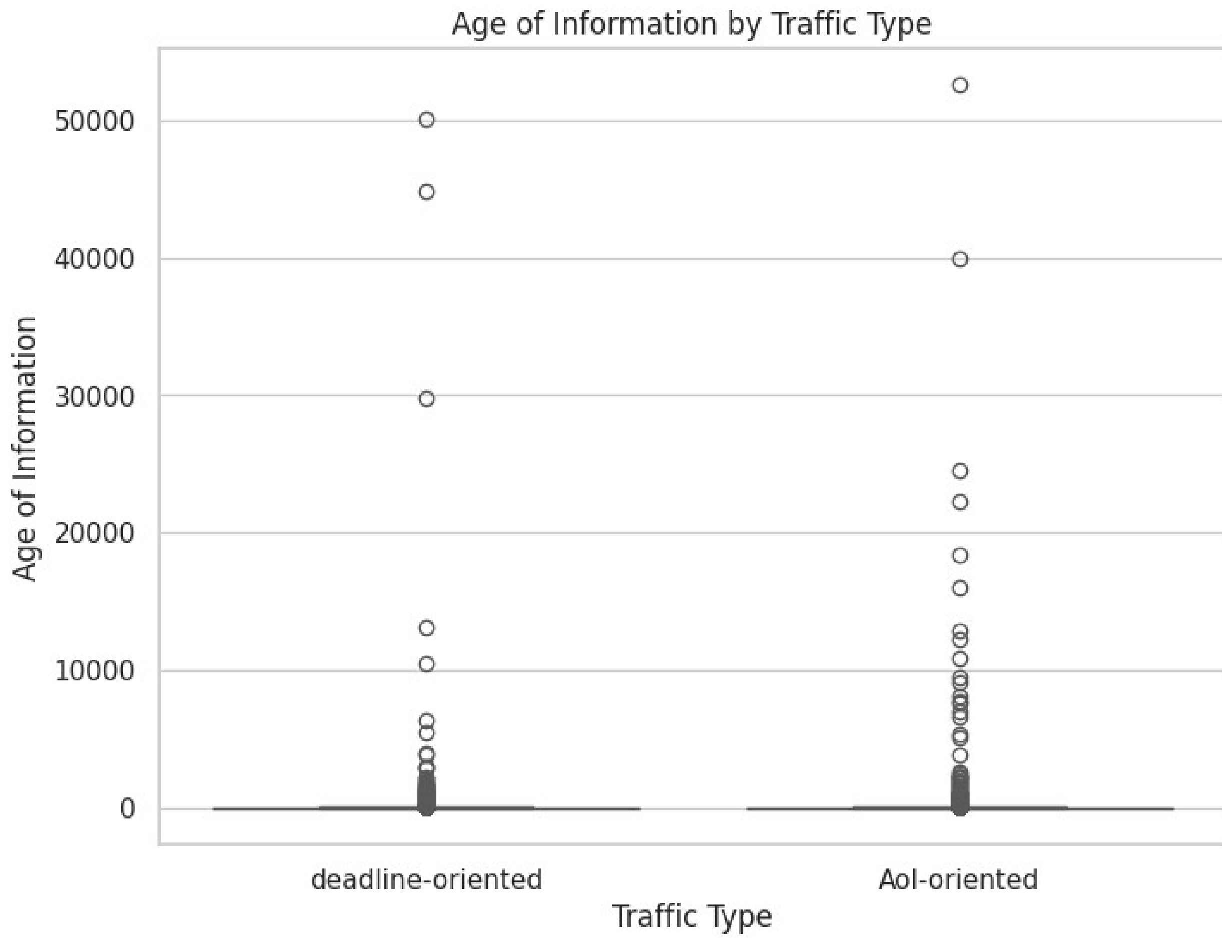
```python
# 1. Scatter plot: transmission_probability vs age_of_information
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='transmission_probability',
y='age_of_information', hue='traffic_type')
```

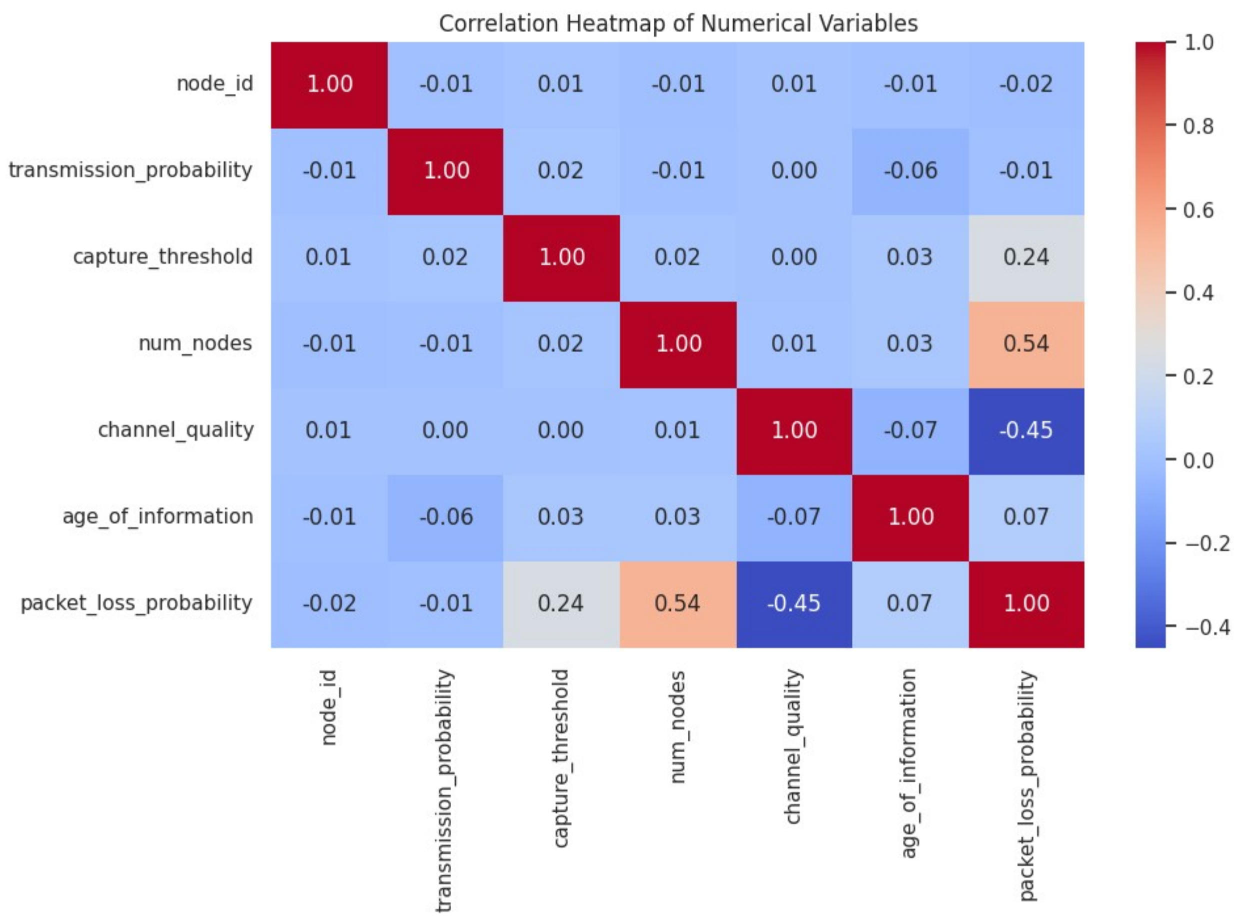
```
plt.title("Transmission Probability vs Age of Information")
plt.xlabel("Transmission Probability")
plt.ylabel("Age of Information")
plt.legend(title='Traffic Type')
plt.show()

# 2. Box plot: age_of_information grouped by traffic_type
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='traffic_type', y='age_of_information',
palette="Set2")
plt.title("Age of Information by Traffic Type")
plt.xlabel("Traffic Type")
plt.ylabel("Age of Information")
plt.show()

# 3. Heatmap: correlation between numerical variables
plt.figure(figsize=(10, 6))
correlation_matrix = df.corr(numeric_only=True)
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Heatmap of Numerical Variables")
plt.show()
```




```
<ipython-input-4-b848a2611701>:12: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=df, x='traffic_type', y='age_of_information',
palette="Set2")
```





1. From the scatter plot, we can observe that higher transmission probabilities generally lead to lower Age of Information (AoI). This suggests that when devices successfully transmit data more often, information stays more up-to-date.

2. We can see from the boxplot that AOI is less urgenet with more spreadout times.

3. for the heatmap age_of_information negatively correlates with transmission_probability. packet_loss_probability positively correlates with age_of_information.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# a) Select relevant features for predicting 'age_of_information'
# Let's assume 'transmission_probability' and 'traffic_type' are
relevant predictors
# If 'traffic_type' is categorical, we need to encode it first
df['traffic_type'] = pd.Categorical(df['traffic_type']).codes  #
Encoding categorical 'traffic_type'

# Select features (X) and target variable (y)
X = df[['transmission_probability', 'traffic_type']]
```

```
y = df['age_of_information']

# b) 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)

# c) Scale the features using StandardScaler
scaler = StandardScaler()

# Fit and transform on training data, and transform on test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display the first few rows of the scaled data (optional)
print("\n□ First few rows of scaled training data:")
print(pd.DataFrame(X_train_scaled, columns=X.columns).head())
```

```
□ First few rows of scaled training data:
   transmission_probability  traffic_type
0                  0.180105     -1.015622
1                  0.527129     -1.015622
2                  0.180105     -1.015622
3                 -1.207990      0.984618
4                  0.527129     -1.015622
```

debugging code

```
# Check for NaN or infinite values in the target variable
print("\n□ Checking for NaN or infinite values in target:")
print(np.isnan(y).sum())  # Check for NaN in target
print(np.isinf(y).sum())  # Check for infinite values in target
```

```
□ Checking for NaN or infinite values in target:
0
1397
```

we'll need to make sure to remove the infinite values in the code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Data Preprocessing
```

```
# Assume 'df' is your DataFrame and 'age_of_information' is the target
# Select relevant features and target variable
X = df[['transmission_probability', 'traffic_type']]  # Example
features
y = df['age_of_information']  # Target variable

# Handle missing and infinite values (if any)
X = X.dropna()  # Dropping rows with missing values in features
y = y[X.index]  # Ensure target variable is aligned with X

# Remove rows where target y has infinite or NaN values
y = y.replace([np.inf, -np.inf], np.nan)
y = y.dropna()  # Remove any rows where y is NaN (after replacing
infinities)

# Ensure X is aligned with y after dropping NaN values from y
X = X.loc[y.index]

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train_scaled, X_test_scaled, y_train, y_test =
train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 2: Hyperparameter Tuning using GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [10, 20, None],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples to split a
node
    'min_samples_leaf': [1, 2, 4]  # Minimum samples required at a
leaf node
}

# Initialize RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
cv=5, n_jobs=-1, scoring='neg_mean_squared_error')
grid_search.fit(X_train_scaled, y_train)

# Output the best parameters found by GridSearchCV
print(f"Best parameters found: {grid_search.best_params_}")

# Step 3: Train the model using the best parameters
```

```
best_rf_model = grid_search.best_estimator_

# Train the model
best_rf_model.fit(X_train_scaled, y_train)

# Step 4: Make predictions on the test set
y_pred = best_rf_model.predict(X_test_scaled)

# Step 5: Evaluate the model using MSE and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Output the evaluation metrics
print(f"□ Mean Squared Error (MSE): {mse:.4f}")
print(f"□ R-squared Score: {r2:.4f}")

----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-21-aa35a19c5bbd> in <cell line: 0>()
     45 # Perform GridSearchCV for hyperparameter tuning
     46 grid_search = GridSearchCV(estimator=rf_model,
param_grid=param_grid, cv=5, n_jobs=-1,
scoring='neg_mean_squared_error')
---> 47 grid_search.fit(X_train_scaled, y_train)
     48
     49 # Output the best parameters found by GridSearchCV

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in
wrapper(estimator, *args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_searc
h.py in fit(self, X, y, **params)
   1022                 return results
   1023
-> 1024             self._run_search(evaluate_candidates)
   1025
   1026             # multimetric is determined here because in the
case of a callable

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_searc
h.py in _run_search(self, evaluate_candidates)
   1569     def _run_search(self, evaluate_candidates):
   1570         """Search all candidates in param_grid"""
```

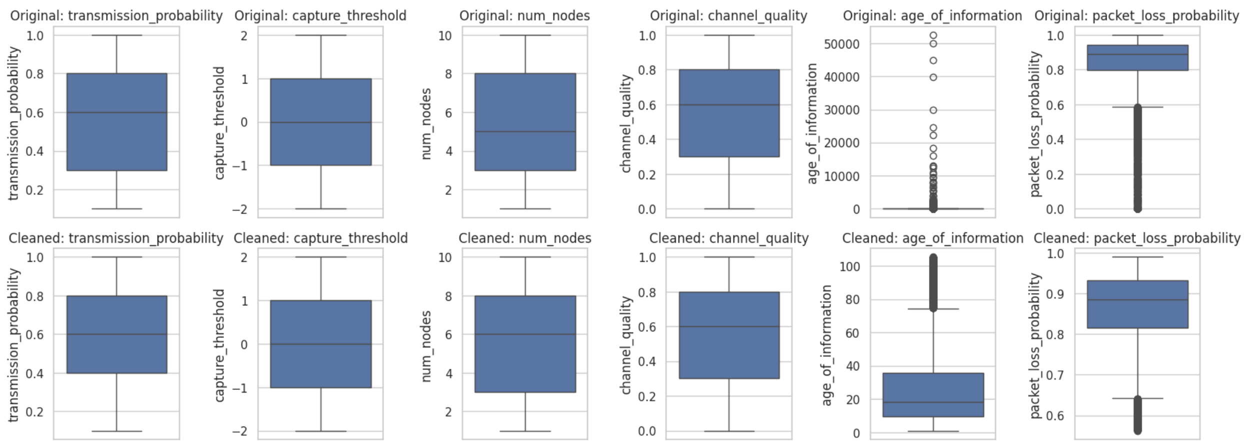
```
-> 1571         evaluate_candidates(ParameterGrid(self.param_grid))
   1572
   1573

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_searc
h.py in evaluate_candidates(candidate_params, cv, more_results)
    968                     )
    969
--> 970                 out = parallel(
    971                     delayed(_fit_and_score)(
    972                         clone(base_estimator),

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in
__call__(self, iterable)
     75             for delayed_func, args, kwargs in iterable
     76         )
---> 77         return super().__call__(iterable_with_config)
     78
     79

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in
__call__(self, iterable)
   2005         next(output)
   2006
-> 2007         return output if self.return_generator else
list(output)
   2008
   2009     def __repr__(self):

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in
_get_outputs(self, iterator, pre_dispatch)
   1648
   1649             with self._backend.retrieval_context():
-> 1650                 yield from self._retrieve()
   1651
   1652         except GeneratorExit:

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in
_retrieve(self)
   1760                 (self._jobs[0].get_status(
   1761                     timeout=self.timeout) == TASK_PENDING)):
-> 1762                 time.sleep(0.01)
   1763                 continue
   1764

KeyboardInterrupt:
```

This had really bad scores. I think the dataset has outliers. (error is cause the runtime takes forever and I needed to restart the kernel)

```python
import numpy as np

# Replace inf and -inf with NaN
df.replace([np.inf, -np.inf], np.nan, inplace=True)

# Drop rows with NaN values (now includes those that were inf)
df.dropna(inplace=True)

numeric_cols = ['age_of_information', 'transmission_probability']  #
Add more if needed
df_cleaned = remove_outliers_iqr(df, numeric_cols)


numeric_cols = [
    'transmission_probability',
    'capture_threshold',
    'num_nodes',
    'channel_quality',
    'age_of_information',
    'packet_loss_probability'
]
df_cleaned = remove_outliers_iqr(df, numeric_cols)

import numpy as np

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

df_cleaned = remove_outliers_iqr(df, numeric_cols)
```

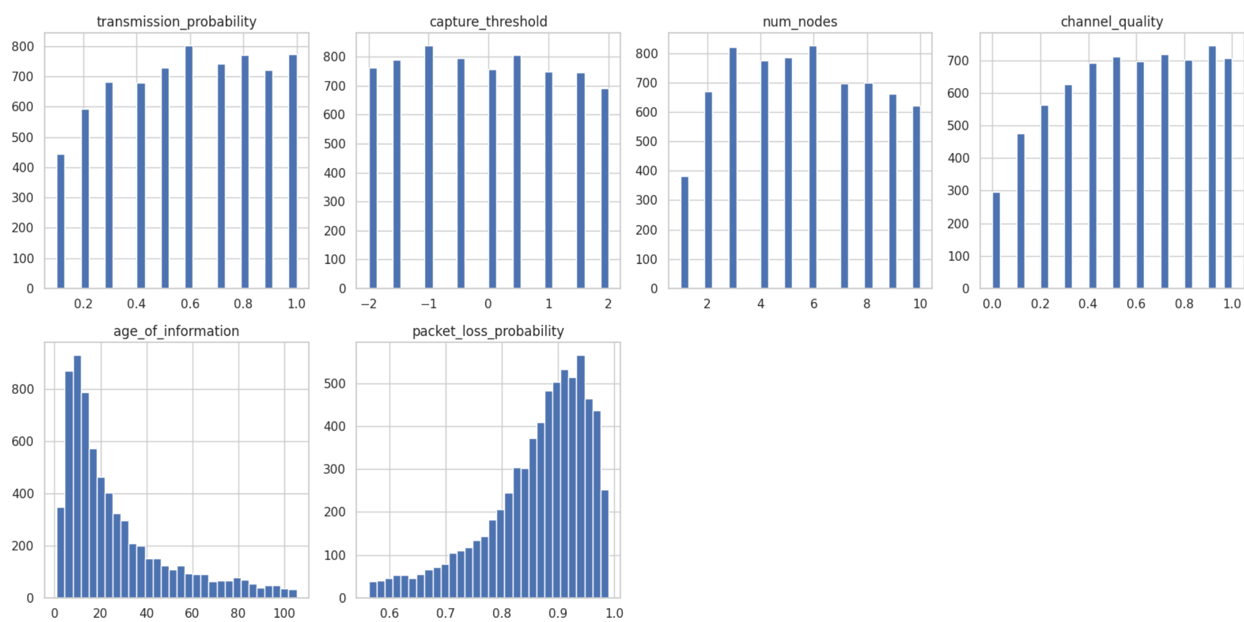
This should get rid of all missing values and outliers.

```python
def plot_boxplots(original_df, cleaned_df, columns):
    plt.figure(figsize=(16, 6))
    for i, col in enumerate(columns):
        plt.subplot(2, len(columns), i+1)
        sns.boxplot(data=original_df, y=col)
        plt.title(f'Original: {col}')

        plt.subplot(2, len(columns), i+1+len(columns))
        sns.boxplot(data=cleaned_df, y=col)
        plt.title(f'Cleaned: {col}')
    plt.tight_layout()
    plt.show()

plot_boxplots(df, df_cleaned, numeric_cols)
```



From here it looks like it was cleaned up pretty well especially aoi

```
def plot_histograms(df, columns):
    df[columns].hist(bins=30, figsize=(16, 8), layout=(2,
int(len(columns)/2)+1))
    plt.tight_layout()
    plt.show()

plot_histograms(df_cleaned, numeric_cols)
```




```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# □ Select relevant features
features = ['transmission_probability', 'capture_threshold',
'num_nodes',
            'channel_quality', 'packet_loss_probability']
```

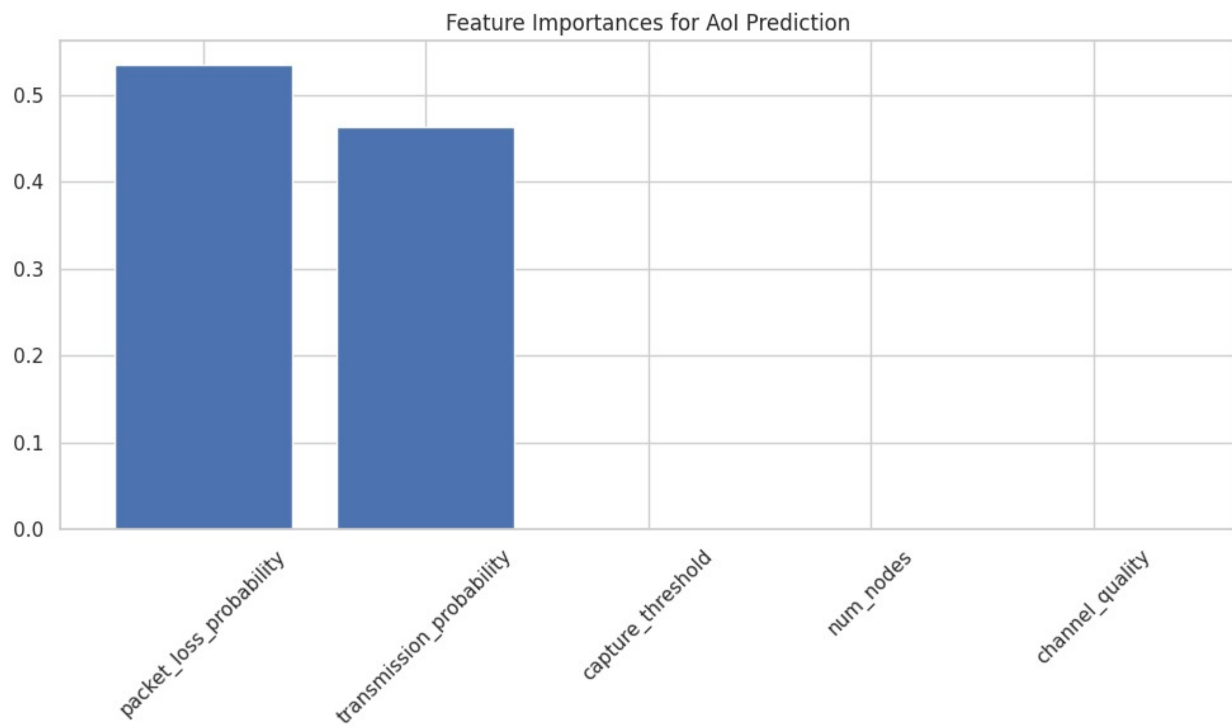
```
target = 'age_of_information'

X = df_cleaned[features]
y = df_cleaned[target]

# □ Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
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)

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# □ Initialize and Train
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

# □ Predict
y_pred = rf.predict(X_test_scaled)

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

print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared Score: {r2:.4f}")
```

```
Mean Squared Error: 1.4772
R-squared Score: 0.9973
```

much better scores.

```
import matplotlib.pyplot as plt
import numpy as np

# Get feature importances
importances = rf.feature_importances_
feature_names = features.columns

# Sort importances
indices = np.argsort(importances)[::-1]

# Plot
plt.figure(figsize=(10, 6))
plt.title("Feature Importances for AoI Prediction")
plt.bar(range(len(importances)), importances[indices], align='center')
```

```
plt.xticks(range(len(importances)), [feature_names[i] for i in
indices], rotation=45)
plt.tight_layout()
plt.show()
```



packet loss probabilty is the most important input.

Step 1: Define 3 Hypothetical Network Configurations

We'll evaluate how the model handles different network conditions by testing the following hypothetical configurations:

Config	Transmission Probability	Capture Threshold (dBm)	Number of Nodes	Channel Quality	Packet Loss Probability
1	0.8	-85	5	0.9	0.05
2	0.5	-70	10	0.6	0.15
3	0.2	-60	15	0.3	0.25

These represent:

- **Config 1**: Optimized network with high reliability
- **Config 2**: Balanced network
- **Config 3**: Challenged network with more interference and nodes

```python
import pandas as pd

# Step 2: Define new configs
new_configs = pd.DataFrame([
    {
        'transmission_probability': 0.8,
        'capture_threshold': -85,
        'num_nodes': 5,
        'channel_quality': 0.9,
        'packet_loss_probability': 0.05
    },
    {
        'transmission_probability': 0.5,
        'capture_threshold': -70,
        'num_nodes': 10,
        'channel_quality': 0.6,
        'packet_loss_probability': 0.15
    },
    {
        'transmission_probability': 0.2,
        'capture_threshold': -60,
        'num_nodes': 15,
        'channel_quality': 0.3,
        'packet_loss_probability': 0.25
    }
])

# Step 3: Scale
new_configs_scaled = scaler.transform(new_configs)

# Step 4: Predict AoI
predicted_aoi = rf.predict(new_configs_scaled)

# Step 5: Show results
for i, aoipred in enumerate(predicted_aoi):
    print(f"Config {i+1} → Predicted AoI: {aoipred:.2f}")
```

```
Config 1 → Predicted AoI: 2.54
Config 2 → Predicted AoI: 3.78
Config 3 → Predicted AoI: 12.59
```

Step 2-5: Model Predictions for Hypothetical Network Configurations

Based on the trained Random Forest model, the predicted **Age of Information (AoI)** for the three hypothetical network configurations is as follows:

Config	Transmission Probability	Capture Threshold (dBm)	Number of Nodes	Channel Quality	Packet Loss Probability	**Predicted AoI**
1	0.8	-85	5	0.9	0.05	**2.54**
2	0.5	-70	10	0.6	0.15	**3.78**
3	0.2	-60	15	0.3	0.25	**12.59**

Interpretation:

- **Config 1**: With high transmission probability and low packet loss, AoI is predicted to be the lowest (2.54), indicating that data freshness is maintained well.
- **Config 2**: As the network balance shifts (lower transmission probability, increased number of nodes, and moderate packet loss), AoI increases slightly to 3.78.
- **Config 3**: With further reduced transmission probability, higher packet loss, and more nodes, AoI increases significantly to 12.59, showing how network challenges affect data freshness.

These predictions reflect the impact of network parameters on the freshness of data and highlight the trade-off between reliability (AoI) and packet loss.

Bonus challenge attempt

```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Prepare the features (X) and target (Y) for AoI and PLP
X = df_cleaned[['transmission_probability', 'capture_threshold',
'num_nodes', 'channel_quality']].values
y = df_cleaned[['age_of_information',
'packet_loss_probability']].values

# 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_dim=X.shape[1]))
# First hidden layer
model.add(layers.Dense(32, activation='relu'))  # Second hidden layer
model.add(layers.Dense(2))  # Output layer (2 outputs: AoI and PLP)
```

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-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)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	320
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 2)	66

```
 Total params: 2,466 (9.63 KB)

 Trainable params: 2,466 (9.63 KB)

 Non-trainable params: 0 (0.00 B)
```

```
from tensorflow.keras import layers, models

# Build the model using the Input layer
model = models.Sequential()
model.add(layers.Input(shape=(X.shape[1],)))  # Explicit Input layer
model.add(layers.Dense(64, activation='relu'))  # First hidden layer
model.add(layers.Dense(32, activation='relu'))  # Second hidden layer
model.add(layers.Dense(2))  # Output layer (2 outputs: AoI and PLP)

# Compile the model
```

```
model.compile(optimizer='adam', loss='mean_squared_error')

# Model summary
model.summary()

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━
━━━━━┩
│ dense_3 (Dense)                  │ (None, 64)              │
320 │
├──────────────────────────────────┼─────────────────────────┼────────
─────┤
│ dense_4 (Dense)                  │ (None, 32)              │
2,080 │
├──────────────────────────────────┼─────────────────────────┼────────
─────┤
│ dense_5 (Dense)                  │ (None, 2)               │
66 │
└──────────────────────────────────┴─────────────────────────┴────────
─────┘

 Total params: 2,466 (9.63 KB)

 Trainable params: 2,466 (9.63 KB)

 Non-trainable params: 0 (0.00 B)

# Assuming 'df_cleaned' is the DataFrame after cleaning
# Selecting relevant features (X) and target columns (y)
X = df_cleaned[['transmission_probability', 'capture_threshold',
'num_nodes', 'channel_quality']]  # Add more features as needed
y = df_cleaned[['age_of_information', 'packet_loss_probability']]  #
Both AoI and PLP as target columns

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

# Train the model on the training data
model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))

Epoch 1/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 9s 26ms/step - loss: 539.3204 - val_loss:
270.7215
```

```
Epoch 2/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 262.3775 - val_loss:
254.9305
Epoch 3/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 246.8925 - val_loss:
231.9937
Epoch 4/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 220.4366 - val_loss:
191.8694
Epoch 5/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 192.4761 - val_loss:
161.2471
Epoch 6/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 152.3928 - val_loss:
151.3628
Epoch 7/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 146.1539 - val_loss:
145.7178
Epoch 8/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 141.2962 - val_loss:
144.7849
Epoch 9/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 135.3309 - val_loss:
142.8228
Epoch 10/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 133.5355 - val_loss:
140.5903
Epoch 11/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 138.8777 - val_loss:
139.5240
Epoch 12/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 128.8521 - val_loss:
141.4290
Epoch 13/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 127.6286 - val_loss:
140.1006
Epoch 14/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 131.6016 - val_loss:
139.9998
Epoch 15/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 122.8927 - val_loss:
136.5833
Epoch 16/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 122.6111 - val_loss:
136.6871
Epoch 17/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 122.0937 - val_loss:
137.8368
Epoch 18/50
```

```
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 122.3432 - val_loss:
135.7424
Epoch 19/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 122.0322 - val_loss:
134.5417
Epoch 20/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 125.4787 - val_loss:
136.6556
Epoch 21/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 119.5465 - val_loss:
134.1103
Epoch 22/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 114.8401 - val_loss:
133.7836
Epoch 23/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 121.3905 - val_loss:
143.4993
Epoch 24/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 117.5178 - val_loss:
134.4767
Epoch 25/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 118.0326 - val_loss:
132.1146
Epoch 26/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 119.6698 - val_loss:
133.9807
Epoch 27/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 127.3684 - val_loss:
138.1160
Epoch 28/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 112.5555 - val_loss:
131.6400
Epoch 29/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 112.8220 - val_loss:
132.7266
Epoch 30/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 113.9262 - val_loss:
131.2306
Epoch 31/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 104.5520 - val_loss:
132.1486
Epoch 32/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 121.9466 - val_loss:
131.6868
Epoch 33/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 111.9186 - val_loss:
131.1000
Epoch 34/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 115.6837 - val_loss:
```

```
130.2233
Epoch 35/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 107.4047 - val_loss:
131.2508
Epoch 36/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 117.7580 - val_loss:
133.3070
Epoch 37/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 111.7815 - val_loss:
133.4898
Epoch 38/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 121.8282 - val_loss:
130.0134
Epoch 39/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 114.9414 - val_loss:
130.9641
Epoch 40/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 117.5985 - val_loss:
131.7112
Epoch 41/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 108.7112 - val_loss:
131.0915
Epoch 42/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 112.5896 - val_loss:
134.2335
Epoch 43/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 114.9179 - val_loss:
131.3852
Epoch 44/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 117.9340 - val_loss:
128.9680
Epoch 45/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 111.4219 - val_loss:
129.3838
Epoch 46/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 115.9307 - val_loss:
130.8157
Epoch 47/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 103.9584 - val_loss:
128.8830
Epoch 48/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 116.0170 - val_loss:
129.3720
Epoch 49/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 115.2308 - val_loss:
136.3721
Epoch 50/50
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 107.0594 - val_loss:
128.6051
```

```
<keras.src.callbacks.history.History at 0x7bd8db849710>
```

note: the epochs got better overtime

```python
# Evaluate the model on the test data
loss = model.evaluate(X_test, y_test)
print(f"Test Loss (Mean Squared Error): {loss}")
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 126.0250
Test Loss (Mean Squared Error): 128.6051483154297
```

```python
from sklearn.metrics import r2_score

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

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print(f"R-squared Score: {r2}")
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
R-squared Score: 0.46397024393081665
```

Model Performance Evaluation

- **Test Loss (MSE)**: 128.61
  The Mean Squared Error (MSE) value indicates the model's error when predicting the Age of Information (AoI) and Packet Loss Probability (PLP) on the test set. Lower values are better. In this case, the model seems to have a moderately low error, but it is important to compare it with the R^2 score for a deeper understanding of its performance.

- **R-squared Score (R^2)**: 0.464
  The R^2 score of **46.4%** suggests that the model is able to explain less than half of the variance in the AoI and PLP predictions. This is relatively low, indicating that the model is not capturing all the relevant patterns in the data. While it performs better than random guessing, there is significant room for improvement.

Analysis and Insights

- **Model Fit**:
  The **R^2 score of 0.464** indicates that the model might be underfitting the data. This means that it is not fully utilizing the available features to make predictions. Additional complexity or data might help improve the performance.

- **Potential Improvements**:
  The model could benefit from the following enhancements:

- **More complex architecture**: Adding more layers, neurons, or experimenting with different activation functions.
- **Feature engineering**: Introducing new features or performing feature selection to remove irrelevant or noisy features.
- **Hyperparameter tuning**: Adjusting parameters like learning rate, batch size, or number of epochs to better optimize the model.
- **More data**: If additional data is available, it could help improve the model's accuracy and generalization.