```
import numpy as np
import pandas as pd
import random
from datetime import datetime, timedelta

# Function to generate timestamp
def generate_timestamp(start, end, num_samples):
    time_diff = (end - start).total_seconds()
    timestamps = [start + timedelta(seconds=random.uniform(0, time_diff)) for _ in range(num_samples)]
    timestamps.sort()
    return timestamps

# Settings
num_samples = 10000
start_time = datetime(2024, 1, 1)
end_time = datetime(2024, 12, 31)
sensor_types = ['temperature', 'vibration', 'pressure']
normal_ranges = {
    'temperature': (20, 80),  # degrees Celsius
    'vibration': (0.1, 5.0),  # mm/s
    'pressure': (100, 500)    # kPa
}

# Generate Data
data = []
timestamps = generate_timestamp(start_time, end_time, num_samples)
for timestamp in timestamps:
    sample = {'timestamp': timestamp}
    for sensor in sensor_types:
        normal_range = normal_ranges[sensor]
        value = random.uniform(*normal_range)
        sample[sensor] = value
    data.append(sample)

# Convert to DataFrame
df = pd.DataFrame(data)

# Save to CSV
df.to_csv('created_iot_data.csv', index=False)

```

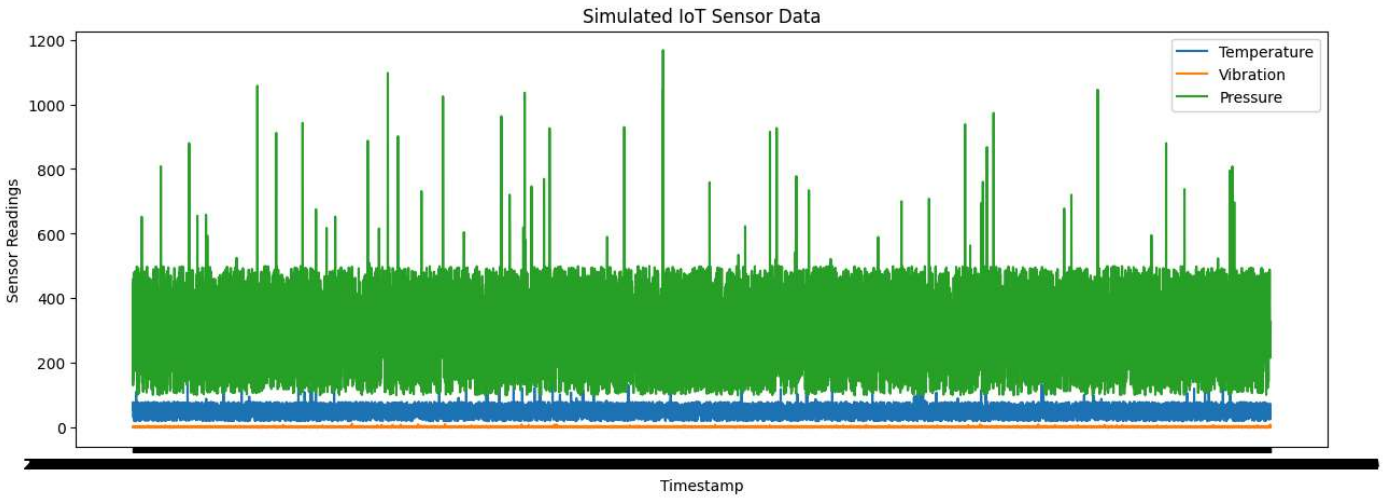
```
print(df)
```

```
                     timestamp  temperature  vibration    pressure
0    2024-01-01 02:42:21.183460    49.392945   1.169905  450.649255
1    2024-01-01 03:16:55.114979    77.083568   2.271737  155.279227
2    2024-01-01 04:04:25.978733    30.959897   4.579936  158.667657
3    2024-01-01 04:25:31.229079    74.834754   3.131460  424.348659
4    2024-01-01 05:09:22.707758    48.665105   3.033871  294.505406
...                         ...          ...        ...         ...
9995 2024-12-30 20:18:22.246433    69.329737   4.262927  332.396657
9996 2024-12-30 20:40:15.261398    57.582948   0.560980  250.020322
9997 2024-12-30 21:29:25.914441    68.883325   3.753144  145.952556
9998 2024-12-30 22:19:45.044972    29.907982   1.498958  333.970891
9999 2024-12-30 23:59:34.998156    64.380688   4.035525  108.546850

[10000 rows x 4 columns]
```

```
def introduce_anomalies(df, anomaly_fraction=0.08):
    num_anomalies = int(len(df) * anomaly_fraction)
    anomaly_indices = random.sample(range(len(df)), num_anomalies)
    for idx in anomaly_indices:
        sensor = random.choice(sensor_types)
        # Create a spike or drop in the sensor reading
        df.at[idx, sensor] = df.at[idx, sensor] * random.uniform(1.5, 2.5)
    return df

df_with_anomalies = introduce_anomalies(df)
df_with_anomalies.to_csv('created_iot_data_with_anomalies.csv', index=False)

```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
```

```
df = pd.read_csv('created_iot_data_with_anomalies.csv')

# Plotting
plt.figure(figsize=(15, 5))
sns.lineplot(data=df, x='timestamp', y='temperature', label='Temperature')
sns.lineplot(data=df, x='timestamp', y='vibration', label='Vibration')
sns.lineplot(data=df, x='timestamp', y='pressure', label='Pressure')
plt.xlabel('Timestamp')
plt.ylabel('Sensor Readings')
plt.legend()
plt.title('Simulated IoT Sensor Data')
plt.show()

```




```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

```
# Load data
df = pd.read_csv('created_iot_data_with_anomalies.csv')

# Convert timestamp to datetime
df['timestamp'] = pd.to_datetime(df['timestamp'])

```

```
print(df)
```

```
                     timestamp  temperature  vibration    pressure
0    2024-01-01 02:42:21.183460    49.392945   1.169905  450.649255
1    2024-01-01 03:16:55.114979   144.194722   2.271737  155.279227
2    2024-01-01 04:04:25.978733    30.959897   4.579936  158.667657
3    2024-01-01 04:25:31.229079    74.834754   3.131460  424.348659
4    2024-01-01 05:09:22.707758    48.665105   3.033871  294.505406
...                         ...          ...        ...         ...
9995 2024-12-30 20:18:22.246433    69.329737   4.262927  332.396657
9996 2024-12-30 20:40:15.261398    57.582948   1.135141  250.020322
9997 2024-12-30 21:29:25.914441    68.883325   3.753144  145.952556
9998 2024-12-30 22:19:45.044972    29.907982   1.498958  333.970891
9999 2024-12-30 23:59:34.998156    64.380688   4.035525  108.546850

[10000 rows x 4 columns]
```

```
# Create labels for maintenance (1 for anomaly, 0 for normal)
df['maintenance_required'] = np.where(
    (df['temperature'] > 80) | (df['vibration'] > 5) | (df['pressure'] > 500), 1, 0)

# Drop timestamp column
```

```
df = df.drop(columns=['timestamp'])

# Features and labels
X = df.drop(columns=['maintenance_required'])
y = df['maintenance_required']
```

```
Start coding or generate with AI.
```

```
      temperature  vibration    pressure  maintenance_required
0       56.152099   1.917923  297.422215                     0
1       78.738909   4.027058  130.785814                     0
2       66.432220   1.984206  440.468128                     0
3       38.158172   0.818703  146.033380                     0
4       30.324411   3.432388  461.625207                     0
...           ...        ...         ...                   ...
9995    39.699535   9.841437  307.510785                     1
9996    59.230216   0.556972  216.407399                     0
9997    26.337234   1.430932  305.655371                     0
9998    38.354017   2.432895  284.487005                     0
9999    42.080552   4.704790  325.507490                     0

[10000 rows x 4 columns]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import accuracy_score, classification_report
```

```
Start coding or generate with AI.
```

Generated code may be subject to a license | aggarwalpiush/bravespace

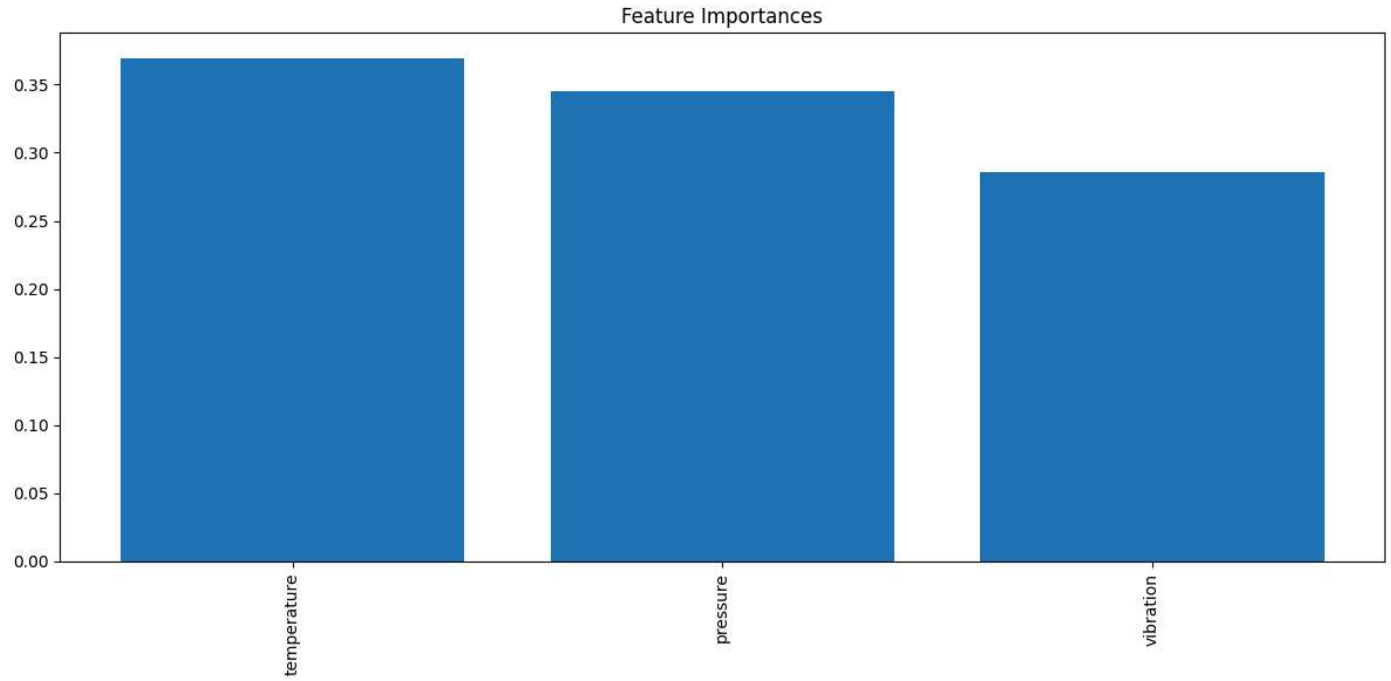
```
# Make predictions
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1972
           1       1.00      1.00      1.00        28

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000

Confusion Matrix:
 [[1972    0]
 [   0   28]]
ROC AUC Score: 1.0
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Model Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
```

```
Classification Model Accuracy: 0.921
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      1860
           1       0.18      0.04      0.06       140

    accuracy                           0.92      2000
   macro avg       0.56      0.51      0.51      2000
weighted avg       0.88      0.92      0.90      2000
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Feature importance
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(12, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices], align='center')
plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)
plt.tight_layout()
plt.show()
```




```
def predict_maintenance(features):

    pred = model.predict([features])

    return {
        'Needs Maintenance' if pred[0] == 1 else 'Normal'
    }

# Example prediction
input_features = [90,4,500]
prediction = predict_maintenance(input_features)
print(prediction)
```

```
{'Prediction': 'Needs Maintenance'}
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but RandomForestClassifie
  warnings.warn(
```

```
Start coding or generate with AI.
```

```
import joblib

# Save the model to a file
joblib.dump(model, 'predict_maintenance_model.pkl')

print("Model saved to 'predict_maintenance_model.pkl'")

```

```
Model saved to 'predict_maintenance_model.pkl'
```