In [1]:
```
import pandas as pd
import numpy as np

dataset = pd.read_csv("intellicath7.csv")
dataset.head
```

Out[1]:
```
<bound method NDFrame.head of       patient_id  time_stamp  urine_output  urine_flow_rate  \
0              1          60    131.125832         2.316438
1              1         120    286.692863         4.884025
2              1         180    227.638364         3.823384
3              1         240    191.637791         3.376382
4              1         300     72.125033         1.082274
...          ...         ...           ...              ...
2495         500          60    162.927880         2.874424
2496         500         120    185.865338         2.945508
2497         500         180    263.705829         4.326234
2498         500         240    294.799623         5.000000
2499         500         300    140.047736         2.373054

      catheter_bag_volume  remaining_catheter_bag_volume  time
0              354.271968                     645.728032  4:27
1              426.092093                     573.907907  1:55
2              769.092654                     230.907346  0:58
3              306.003947                     693.996053  3:19
4              782.684716                     217.315284  3:03
...                   ...                            ...   ...
2495            70.504396                     929.495604  5:12
2496            65.317318                     934.682682  5:07
2497           765.285937                     234.714063  0:52
2498           191.565077                     808.434923  2:38
2499           601.877929                     398.122071  2:40

[2500 rows x 7 columns]>
```

In [2]:
```
X = dataset[[
    "urine_output",
    "urine_flow_rate",
    "catheter_bag_volume",
    "remaining_catheter_bag_volume"
]]
y = dataset["time"]
```

In [3]:
```
def time_to_decimal(hours):
    try:
        if isinstance(hours, str) and ":" in hours:
            h, m = map(int, hours.split(":"))
            return h + m / 60
        else:
            return 0
    except ValueError:
        return 0
if "time" in dataset.columns:
    dataset["time"] = dataset["time"].fillna("00:00")  # Replace NaN with "00:00"
    dataset["Time"] = dataset["time"].apply(time_to_decimal)
    y_decimal = dataset["Time"]
else:
    y_decimal = y.apply(time_to_decimal)
```

In [4]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_poly)
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_decimal, test_size=0.2, random_state=42)

model_1 = Ridge(alpha=1.0)
model_1.fit(X_train, y_train)
```

Out[5]:
```
Ridge()
```

In [6]:
```
y_pred = model_1.predict(X_test)
```

In [9]:
```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
cv_scores = cross_val_score(model_1, X_scaled, y_decimal, cv=5, scoring='r2')
mean_cv_r2 = np.mean(cv_scores)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R2 Score (Test Set):", r2)
print("Mean R2 Score (Cross-Validation):", mean_cv_r2)
print("Coefficients:", model_1.coef_)
print("Intercept:", model_1.intercept_)
```

```
Mean Squared Error (MSE): 1.5590203779964005
Mean Absolute Error (MAE): 0.6594906204435189
R2 Score (Test Set): 0.8811568893092434
Mean R2 Score (Cross-Validation): 0.8655371459305788
Coefficients: [-1.040657   -3.31754024 -1.02912667  1.02912667  1.13505027  1.85185377
  0.48405751 -1.68439187  2.55758747 -1.52804137 -2.65199576 -1.11138719
  0.02890721  0.91096374]
Intercept: 4.35896466576585
```

In [10]:
```
from joblib import dump

dump(model_1, 'model_1.joblib')
```

Out[10]:
```
['model_1.joblib']
```

In [11]:
```
from joblib import load

model_loaded = load('model_1.joblib')

prediction = model_loaded.predict(X_test)
```

In [13]:
```
import pickle

with open("model_1.pkl", "wb") as file:
    pickle.dump(model_1, file)

# Save the polynomial features and scaler for future use
with open("polynomial_features.pkl", "wb") as file:
    pickle.dump(poly, file)

with open("scaler.pkl", "wb") as file:
    pickle.dump(scaler, file)

model = pickle.load(open('model_1.pkl', 'rb'))
```

In [ ]:
```
```