Creating a Dataset

```
import numpy as np
import pandas as pd
```

```
# Set dataset size
n = 1000  # Number of samples

# Seed for reproducibility
np.random.seed(0)
```

```
# Demographic Features

age = np.random.normal(40, 10, n).astype(int)                                                    # Age around 40 years
employment_status = np.random.choice(['employed', 'self-employed', 'unemployed'], size=n, p=[0.6, 0.2, 0.2])
marital_status = np.random.choice(['single', 'married', 'divorced'], size=n, p=[0.5, 0.4, 0.1])
```

```
# Financial Attributes with values rounded to 3 decimal places

income = np.round(np.random.normal(50000, 15000, n), 2)            # Income with a mean of 50,000
debt_amount = np.round(np.random.normal(20000, 10000, n), 2)       # Debt amount with mean of 20,000
credit_card_balance = np.round(np.random.normal(5000, 2500, n), 2) # Credit card balance with mean of 5000
credit_utilization = np.round(np.random.uniform(0, 1, n), 2)       # Utilization between 0 and 1
```

```
# Behavioral Attributes
late_payments = np.random.poisson(1, n)                                          # Average of 1 late payment
payment_frequency = np.random.choice(['monthly', 'quarterly'], size=n, p=[0.8, 0.2])
average_payment_delay = np.round(np.random.normal(5, 2, n), 2)                             # Payment delay with a mean of 5 days
```

```
# Derived Ratios
debt_to_income_ratio = np.round(debt_amount / (income + 1e-5), 2)            # To avoid division by zero
credit_to_income_ratio = np.round(credit_card_balance / (income + 1e-5), 2)
```

```
creditworthy = ((debt_to_income_ratio < 0.3) &
                (credit_utilization < 0.5) &
                (late_payments < 2)).astype(int)         # 1 for creditworthy, 0 for not
```

```
df = pd.DataFrame({
    'age': age,
    'employment_status': employment_status,
    'marital_status': marital_status,
    'income': income,
    'debt_amount': debt_amount,
    'credit_card_balance': credit_card_balance,
    'credit_utilization': credit_utilization,
    'late_payments': late_payments,
    'payment_frequency': payment_frequency,
    'average_payment_delay': average_payment_delay,
    'debt_to_income_ratio': debt_to_income_ratio,
    'credit_to_income_ratio': credit_to_income_ratio,
    'creditworthy': creditworthy
})
```

```
df.head()
```

	age	employment_status	marital_status	income	debt_amount	credit_card_balance	credit_utilization	late_payments	payment_frequency	averag
0	57	unemployed	divorced	31004.86	19081.89	2326.31	0.69	1	quarterly	
1	44	self-employed	single	49077.58	26512.10	11578.69	0.07	2	monthly	
2	49	unemployed	married	29153.28	21967.70	1467.56	0.69	1	monthly	
3	62	unemployed	married	20694.82	29696.24	6268.58	0.41	0	monthly	
4	58	self-employed	single	54436.69	2813.50	-409.16	0.14	0	quarterly	

Data Processing

```
df.head()
```

	age	employment_status	marital_status	income	debt_amount	credit_card_balance	credit_utilization	late_payments	payment_frequency	averag
0	57	unemployed	divorced	31004.86	19081.89	2326.31	0.69	1	quarterly	
1	44	self-employed	single	49077.58	26512.10	11578.69	0.07	2	monthly	
2	49	unemployed	married	29153.28	21967.70	1467.56	0.69	1	monthly	
3	62	unemployed	married	20694.82	29696.24	6268.58	0.41	0	monthly	
4	58	self-employed	single	54436.69	2813.50	-409.16	0.14	0	quarterly	

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
# Convert categorical features to numeric using Label Encoding
df['employment_status'] = LabelEncoder().fit_transform(df['employment_status'])
df['marital_status'] = LabelEncoder().fit_transform(df['marital_status'])
df['payment_frequency'] = LabelEncoder().fit_transform(df['payment_frequency'])
```

```
# Separate features and target
X = df.drop('creditworthy', axis=1)  # Features
y = df['creditworthy']               # Target variable
```

```
# Scale the feature set to standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Model Selection and Training

```
df.head()
```

	age	employment_status	marital_status	income	debt_amount	credit_card_balance	credit_utilization	late_payments	payment_frequency	averag
0	57	2	0	31004.86	19081.89	2326.31	0.69	1	1	
1	44	1	2	49077.58	26512.10	11578.69	0.07	2	0	
2	49	2	1	29153.28	21967.70	1467.56	0.69	1	0	
3	62	2	1	20694.82	29696.24	6268.58	0.41	0	0	
4	58	1	2	54436.69	2813.50	-409.16	0.14	0	1	

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=0)
```

```
#This command shows the order pair of test and train

print("shape of X_train: ", X_train.shape)
print("shape of Y_train: ", y_train.shape)
print("shape of X_test: ", X_test.shape)
print("shape of Y_test: ", y_test.shape)
```

```
shape of X_train:  (800, 12)
shape of Y_train:  (800,)
shape of X_test:  (200, 12)
shape of Y_test:  (200,)
```

```
# Initialize and train the model
model = RandomForestClassifier(n_estimators=100, random_state=0)
model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=0)
```

```
# Predict the target for the test set
y_pred = model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Model Accuracy:", accuracy)
```

```
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Model Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       177
           1       1.00      1.00      1.00        23

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200


Confusion Matrix:
 [[177   0]
 [  0  23]]
```

```
y_pred = model.predict(X)
score = accuracy_score(y, y_pred)
accuracy = score*100
print(accuracy)
```

```
87.9
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X has feature names, but RandomForestClassifier was fitted without featu
  warnings.warn(
```

Start coding or generate with AI.