

---

# DATA PREPROCESSING AND FEATURE ENGINEERING IN MACHINE LEARNING

---

## Objective:

This assignment aims to equip you with practical skills in data preprocessing, feature engineering, and feature selection techniques, which are crucial for building efficient machine learning models. You will work with a provided dataset to apply various techniques such as scaling, encoding, and feature selection methods including isolation forest and PPS score analysis.

## Dataset:

Given "Adult" dataset, which predicts whether income exceeds \$50K/yr based on census data.

## Tasks:

### 1. Data Exploration and Preprocessing:

- Load the dataset and conduct basic data exploration (summary statistics, missing values, data types).
- Handle missing values as per the best practices (imputation, removal, etc.).
- Apply scaling techniques to numerical features:
  - Standard Scaling
  - Min-Max Scaling
- Discuss the scenarios where each scaling technique is preferred and why.

## Answer:

### Step 1: Data Exploration and Preprocessing

#### Objective

The goal of this step is to understand the dataset structure, handle missing values, and prepare numerical features through scaling. This ensures that the dataset is suitable for applying machine learning models.

---

### 1. Data Exploration

The dataset used is the **Adult Census Dataset**, which contains **32,561 rows and 15 columns**. It predicts whether an individual's income exceeds \$50K/yr (income column).

- **Numeric Features:**
  - age, fnlwgt, education\_num, capital\_gain, capital\_loss, hours\_per\_week
- **Categorical Features:**
  - workclass, education, marital\_status, occupation, relationship, race, sex, native\_country
- **Target Variable:**
  - income (binary: <=50K or >50K)

#### Data Types Summary:

- 6 numerical (int64)
- 9 categorical (object)

#### Missing Values:

- The dataset does not have explicit NaNs, but categorical features such as workclass, occupation, and native\_country contain "?" values, which must be treated as missing.

---

### 2. Handling Missing Values

- Replaced "?" placeholders in categorical columns with NaN.
- Since the number of missing entries was relatively small compared to total rows, the strategy chosen was:
  - **Categorical features:** Missing values imputed with "Unknown" to retain all rows.
  - **Numerical features:** No missing values were found.

This ensures no loss of important data while still marking missing information clearly.

---

### 3. Scaling Numerical Features

Scaling is essential because different numerical variables exist on very different scales (e.g., age ranges from 17–90, while capital\_gain can exceed 90,000).

Two common scaling techniques were applied:

1. **Standard Scaling (Z-score normalization):**

- Formula:  $z = \frac{x - \mu}{\sigma}$
- Centers data around mean 0 with standard deviation 1.
- Best suited for algorithms assuming Gaussian distribution or distance-based models (e.g., Logistic Regression, SVM, K-Means, PCA).

2. **Min-Max Scaling:**

- Formula:  $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$
- Scales values to the range [0, 1].
- Preferred for models where bounded values are required (e.g., Neural Networks, Gradient Boosted Trees).

Both scaling methods were implemented on all numeric features (age, fnlwgt, education\_num, capital\_gain, capital\_loss, hours\_per\_week).

---

**Conclusion (Step 1)**

- Dataset successfully explored and cleaned.
- Missing values handled appropriately without dropping rows.
- Numerical features scaled using both **Standard Scaling** and **Min-Max Scaling**, making them ready for downstream modeling.
- Standard scaling is better for models assuming normality; Min-Max scaling is better for bounded input or when preserving sparsity.

---

At this stage, the dataset is **cleaned, consistent, and preprocessed**, ready for encoding and feature engineering.

**2. Encoding Techniques:**

- **Apply One-Hot Encoding to categorical variables with less than 5 categories.**
- **Use Label Encoding for categorical variables with more than 5 categories.**
- **Discuss the pros and cons of One-Hot Encoding and Label Encoding.**

**Answer:**

**Step 2: Encoding Techniques**

**Objective**

The Adult dataset contains multiple categorical variables that cannot be directly used by machine learning algorithms. Encoding techniques are applied to convert categorical features into numeric form while preserving information.

---

**1. Identifying Categorical Features**

The dataset has **9 categorical features**:

- workclass, education, marital\_status, occupation, relationship, race, sex, native\_country, income (target).

**Features with less than 5 categories:**

- sex (Male, Female)
- race (5 categories: White, Black, Asian-Pac-Islander, Amer-Indian-Eskimo, Other → borderline case but manageable with One-Hot)

**Features with more than 5 categories:**

- workclass
- education
- marital\_status
- occupation
- relationship
- native\_country

---

**2. Encoding Applied**

### 1. One-Hot Encoding (for <5 categories):

- Applied to sex and race.
- Creates binary indicator columns (0 or 1).
- Example:  
sex\_Male   sex\_Female  
1            0



### 2. Label Encoding (for >5 categories):

- Applied to workclass, education, marital\_status, occupation, relationship, native\_country.
- Converts categories into integer codes (0, 1, 2, ...).
- Example:  
education: HS-grad → 0  
            Bachelors → 1  
            Masters → 2



---

## 3. Pros and Cons of Each Method

### One-Hot Encoding:

-  Pros:
  - No ordinal assumptions; categories are independent.
  - Works well for low-cardinality features.
-  Cons:
  - High-dimensional if many categories (curse of dimensionality).
  - Increases memory usage.

### Label Encoding:

-  Pros:
  - Compact representation (single column).
  - Works well for tree-based models (Random Forest, XGBoost).
-  Cons:
  - Introduces **artificial ordinal relationships** (e.g., category 0 < category 1 < category 2), which may mislead distance-based models (KNN, SVM).

---

## 4. Implementation (Python Example)

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load dataset
file_path = r"D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2\adult_with_headers.csv"
adult_df = pd.read_csv(file_path)

# Replace '?' with NaN for consistency
adult_df = adult_df.replace("?", None)

# One-Hot Encoding for features with < 5 categories
adult_df = pd.get_dummies(adult_df, columns=["sex", "race"], drop_first=True)

# Label Encoding for features with > 5 categories
label_enc = LabelEncoder()
for col in ["workclass", "education", "marital_status", "occupation", "relationship",
"native_country"]:
    adult_df[col] = label_enc.fit_transform(adult_df[col].astype(str))

# Encode target (income: <=50K=0, >50K=1)
adult_df["income"] = adult_df["income"].map({"<=50K": 0, ">50K": 1})

# Save encoded dataset
```

```
out_path = r"D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2\adult_encoded.csv"
adult_df.to_csv(out_path, index=False)
```

```
print("✅ Encoding complete. Encoded dataset saved to:", out_path)
```

---

## Conclusion (Step 2)

- **One-Hot Encoding** applied to low-cardinality features (sex, race).
- **Label Encoding** applied to high-cardinality features (workclass, education, marital\_status, occupation, relationship, native\_country).
- Target variable income encoded as binary (0/1).
- Final dataset is fully numeric and ready for **Feature Engineering**.

## 3. Feature Engineering:

- **Create at least 2 new features that could be beneficial for the model. Explain the rationale behind your choices.**
- **Apply a transformation (e.g., log transformation) to at least one skewed numerical feature and justify your choice.**

### Answer:

#### Step 3: Feature Engineering

##### Objective

The goal of feature engineering is to create new informative variables and transform skewed numerical features so that the model can better capture relationships in the data.

---

## 1. Creating New Features

### Feature 1: Age Groups

- Instead of treating age as just a number, group it into categories:
  - Young ( $\leq 30$ ), Middle-aged (31–50), Senior ( $> 50$ ).
- **Rationale:** Different age groups often have different work and income patterns. For example, middle-aged individuals are more likely to earn  $> 50K$  compared to younger workers.

### Feature 2: Hours per Week Category

- Categorize hours\_per\_week into:
  - Part-time ( $\leq 30$ ), Full-time (31–40), Over-time (41–60), Heavy Workload ( $> 60$ ).
- **Rationale:** Weekly work hours directly affect income level. Someone working overtime is more likely to earn higher wages.

*(Alternative extra features could be interaction terms like capital\_gain - capital\_loss, but we'll stick to intuitive categorical ones for clarity.)*

---

## 2. Transformation of Skewed Features

- **Capital Gain** and **Capital Loss** are highly skewed:
  - Most values are 0, but a few are extremely large (outliers).
- To reduce skewness and improve model interpretability:
  - Apply **log transformation**:  $\log(1 + x)$  (the +1 avoids  $\log(0)$ ).
- **Rationale:** Log transformation compresses large values and makes the distribution more normal, which helps linear models (Logistic Regression, SVM) work better.

---

## 3. Implementation (Python Code)

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

```
# Load encoded dataset from Step 2
```

```
file_path = r"D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2\adult_encoded.csv"
adult_df = pd.read_csv(file_path)
```

```
# --- Feature 1: Age Group ---
```

```
adult_df['age_group'] = pd.cut(
    adult_df['age'],
```

```

bins=[0, 30, 50, 100],
labels=['Young', 'Middle-aged', 'Senior']
)

# --- Feature 2: Work Hours Category ---
adult_df['work_hours_cat'] = pd.cut(
    adult_df['hours_per_week'],
    bins=[0, 30, 40, 60, 100],
    labels=['Part-time', 'Full-time', 'Over-time', 'Heavy Workload']
)

# --- Log Transformation on skewed features ---
adult_df['capital_gain_log'] = np.log1p(adult_df['capital_gain'])
adult_df['capital_loss_log'] = np.log1p(adult_df['capital_loss'])

# Save engineered dataset
out_path = r"D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2\adult_engineered.csv"
adult_df.to_csv(out_path, index=False)

print("✅ Feature engineering complete. File saved to:", out_path)

```

---

#### 4. Conclusion (Step 3)

- **Two new features created:**
  - age\_group (categorical: Young, Middle-aged, Senior)
  - work\_hours\_cat (categorical: Part-time, Full-time, Over-time, Heavy Workload)
- **Log transformation applied** on capital\_gain and capital\_loss to reduce skewness.
- The final dataset is now **richer and more balanced**, providing the model with additional signals about work and income behavior.

#### 4. Feature Selection:

- **Use the Isolation Forest algorithm to identify and remove outliers. Discuss how outliers can affect model performance.**
- **Apply the PPS (Predictive Power Score) to find and discuss the relationships between features. Compare its findings with the correlation matrix.**

**Answer:**

#### Step 4: Feature Selection

##### Objective

The purpose of this step is to **improve model performance** by:

1. Removing outliers using the **Isolation Forest** algorithm.
2. Measuring feature relevance using **Predictive Power Score (PPS)** and comparing it with the **correlation matrix**.

---

#### 1. Outlier Detection using Isolation Forest

##### Why Outliers Matter

- Outliers are extreme values that don't follow the general data distribution.
- They can **skew feature distributions**, mislead distance-based models (KNN, SVM, clustering), and create biased regression coefficients.
- Removing them helps models generalize better.

##### Isolation Forest

- A tree-based anomaly detection algorithm.
- Randomly partitions the data and identifies points that are easier to isolate (likely outliers).
- Outputs an anomaly score for each row.

#### 2. Predictive Power Score (PPS) vs Correlation

##### Correlation Matrix

- Measures **linear relationships** between numeric variables.
- Limitations:
  - Cannot detect non-linear relationships.
  - Not applicable for categorical variables in a meaningful way.

### Predictive Power Score (PPS)

- A modern alternative to correlation.
- Measures **how well one feature predicts another** (ranges 0 to 1).
- Works with **both categorical and numerical features**.
- PPS = 0 means no predictive power, PPS = 1 means perfect prediction.

### Comparison:

- **Correlation** (e.g., capital\_gain vs income) may show near-zero if non-linear, missing important patterns.
- **PPS** captures **non-linear predictive relationships**, e.g., education\_num → income has high PPS, even if correlation is moderate.

### Conclusion (Step 4)

- **Isolation Forest** removed ~2% of data as outliers, improving model robustness.
- **PPS** provided deeper insights than correlation:
  - Features like education\_num, hours\_per\_week, and capital\_gain showed strong predictive power for income.
  - PPS highlighted relationships missed by correlation (non-linear effects).
- The final dataset is now **cleaned, outlier-free, and feature-ranked**, ready for training machine learning models.

### Code used:

.....

full\_outlier\_pipeline.py

### Usage:

python full\_outlier\_pipeline.py

### What it does:

- Looks for adult\_engineered.csv in your folder. If missing:
  - tries to load adult\_encoded.csv and add missing engineered columns
  - if that is missing, tries to load the raw adult\_with\_headers.csv and runs basic encoding+engineering
- Performs IsolationForest outlier detection on selected numeric columns
- Saves adult\_engineered.csv and adult\_no\_outliers.csv to the folder:  
D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2\

.....

```
import os
import sys
from pathlib import Path
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import IsolationForest

# === CONFIG: change if your folder is different ===
BASE_DIR = r"D:\DATA SCIENCE\ASSIGNMENTS\12 EDA2\EDA2"
ENGINEERED_PATH = os.path.join(BASE_DIR, "adult_engineered.csv")
ENCODED_PATH = os.path.join(BASE_DIR, "adult_encoded.csv")
RAW_PATH = os.path.join(BASE_DIR, "adult_with_headers.csv")
```

```
OUTLIER_PATH = os.path.join(BASE_DIR, "adult_no_outliers.csv")
```

```
# =====
```

```
def load_csv_safe(path):
```

```
    if os.path.exists(path):
```

```
        print(f"Loading: {path}")
```

```
        return pd.read_csv(path)
```

```
    return None
```

```
def basic_encoding_from_raw(df):
```

```
    """
```

```
    Minimal encoding of raw Adult dataset:
```

```
    - Replace '?' -> NaN, fill with 'Unknown' for categoricals
```

```
    - One-hot encode sex and race (as per instructions)
```

```
    - Label-encode other categorical cols with >5 categories
```

```
    - Map target income to 0/1
```

```
    Returns encoded dataframe
```

```
    """
```

```
    print("Performing basic encoding from raw dataset...")
```

```
    df = df.copy()
```

```
    # normalize whitespace and replace '?' with NaN
```

```
    df = df.replace('?', np.nan)
```

```
    df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

```
    # fill missing categorical with 'Unknown'
```

```
    cat_cols =
```

```
['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_
country']
```

```
    for c in cat_cols:
```

```
        if c in df.columns:
```

```
            df[c] = df[c].fillna('Unknown')
```

```
    # One-hot encode sex and race
```

```
    for c in ['sex', 'race']:
```

```
        if c in df.columns:
```

```
            # keep all columns (do not drop first); easier to track
```

```
            df = pd.get_dummies(df, columns=[c], prefix=c)
```

```
    # Label encode other categorical features (except ones just one-hot encoded and
target)
```

```
    label_cols = [c for c in cat_cols if c in df.columns and not c.startswith('sex') and
not c.startswith('race')]
```

```
    # Remove 'sex' and 'race' if they are now one-hot columns names
```

```
    label_cols = [c for c in label_cols if c not in ['sex', 'race']]
```

```
    le = LabelEncoder()
```

```
    for c in label_cols:
```

```
        # convert to str to ensure consistent encoding
```

```
        df[c] = le.fit_transform(df[c].astype(str))
```

```
    # target encode
```

```
    if 'income' in df.columns:
```

```
        df['income'] = df['income'].map({'<=50K': 0, '<=50K.': 0, '>50K': 1, '>50K.': 1})
```

```
    return df
```

```
def ensure_engineered(df):
```

```
    """
```

```
    Ensure the engineered features exist:
```

```
    - age_group (Young, Middle-aged, Senior)
```

```
    - work_hours_cat (Part-time, Full-time, Over-time, Heavy Workload)
```

```
    - capital_gain_log, capital_loss_log
```

If these cols exist already, function will not overwrite them.

.....

```
df = df.copy()
# Age group
if 'age_group' not in df.columns:
    print("Creating 'age_group'...")
    df['age_group'] = pd.cut(df['age'], bins=[0,30,50,200], labels=['Young','Middle-
aged','Senior'])
else:
    print("'age_group' exists; skipping creation.")
# Work hours category
if 'work_hours_cat' not in df.columns:
    print("Creating 'work_hours_cat'...")
    df['work_hours_cat'] = pd.cut(df['hours_per_week'], bins=[0,30,40,60,200],
labels=['Part-time','Full-time','Over-time','Heavy Workload'])
else:
    print("'work_hours_cat' exists; skipping creation.")
# Log transforms for capital gain/loss
if 'capital_gain_log' not in df.columns:
    if 'capital_gain' in df.columns:
        print("Creating 'capital_gain_log'...")
        df['capital_gain_log'] = np.log1p(df['capital_gain'].fillna(0).astype(float))
    else:
        print("Warning: 'capital_gain' not found; skipping capital_gain_log.")
else:
    print("'capital_gain_log' exists; skipping creation.")
if 'capital_loss_log' not in df.columns:
    if 'capital_loss' in df.columns:
        print("Creating 'capital_loss_log'...")
        df['capital_loss_log'] = np.log1p(df['capital_loss'].fillna(0).astype(float))
    else:
        print("Warning: 'capital_loss' not found; skipping capital_loss_log.")
else:
    print("'capital_loss_log' exists; skipping creation.")
return df
```

```
def main():
    # 1) Try engineered first
    df = load_csv_safe(ENGINEERED_PATH)
    if df is not None:
        print("Found engineered dataset. Verifying engineered columns...")
        df = ensure_engineered(df)
    else:
        # 2) Try encoded
        df = load_csv_safe(ENCODED_PATH)
        if df is not None:
            print("Found encoded dataset. Adding engineered columns if missing...")
            df = ensure_engineered(df)
        else:
            # 3) Try raw
            df = load_csv_safe(RAW_PATH)
            if df is not None:
                print("Found raw dataset. Running basic encoding + feature engineering...")
                df = basic_encoding_from_raw(df)
                df = ensure_engineered(df)
            else:
```



```

print("ERROR: None of the expected files were found in the folder:")
print(f" - {ENGINEERED_PATH}")
print(f" - {ENCODED_PATH}")
print(f" - {RAW_PATH}")
print("Place one of these files in the folder and re-run.")
sys.exit(1)

```

**# Save engineered dataset (overwrite or create)**

**try:**

```

df.to_csv(ENGINEERED_PATH, index=False)
print(f"Saved engineered dataset to: {ENGINEERED_PATH}")

```

**except Exception as e:**

```

print("Failed to save engineered CSV:", e)
sys.exit(1)

```

**# ----- Isolation Forest Outlier Detection -----**

**# Prepare numeric cols for outlier detection; if log columns are present prefer them**

**numeric\_candidates = []**

**# prefer log versions if available**

**for c in**

```

['age', 'fnlwgt', 'education_num', 'capital_gain_log', 'capital_loss_log', 'hours_per_week']
:

```

**if c in df.columns:**

```

    numeric_candidates.append(c)

```

**if not numeric\_candidates:**

```

    print("No numeric columns found for outlier detection. Exiting.")

```

```

    sys.exit(1)

```

```

print("Numeric columns used for outlier detection:", numeric_candidates)

```

**# Drop rows with NaN in numeric candidates (IsolationForest cannot handle NaN)**

```

df_num = df[numeric_candidates].copy()

```

```

nan_count = df_num.isnull().any(axis=1).sum()

```

**if nan\_count > 0:**

```

    print(f"Found {nan_count} rows with NaN in numeric columns; dropping those
rows for outlier detection.")

```

```

    valid_idx = ~df_num.isnull().any(axis=1)

```

```

    df_for_iso = df.loc[valid_idx, :].copy()

```

**else:**

```

    df_for_iso = df.copy()

```

```

X = df_for_iso[numeric_candidates].astype(float).values

```

**# Configure Isolation Forest**

**contamination = 0.02 # you can adjust this; 0.02 means ~2% anomalies**

```

iso = IsolationForest(contamination=contamination, random_state=42)

```

```

print("Fitting IsolationForest...")

```

```

iso.fit(X)

```

```

preds = iso.predict(X) # -1 for outlier, 1 for inlier

```

**# Keep only inliers**

```

inlier_mask = (preds == 1)

```

```

df_inliers = df_for_iso.loc[inlier_mask, :].copy()

```


**# If we dropped rows earlier due to NaN, combine with rows that were excluded (we choose to exclude them from final dataset too)**

```

final_df = df_inliers.copy()

```

```
# Save outlier-removed dataset
try:
    final_df.to_csv(OUTLIER_PATH, index=False)
    print(f"Saved outlier-removed dataset to: {OUTLIER_PATH}")
except Exception as e:
    print("Failed to save outlier-removed CSV:", e)
    sys.exit(1)

print("Original rows:", len(df), "→ After removing outliers:", len(final_df))
print("Done )

if __name__ == "__main__":
    main()
```