Steps:

1. Read the data
2. Drop useless columns

In the same time checking nulls in numeric data

1. Data Splitting

I would usually do this step after removing missing values from the data right before the encoding but anyway.

1. Dropping Duplicates
2. Handling Missing Values

* We already checked for missing values in numeric data in step 2
* If we printed a slice from the data, or even opened its Excel sheet, we can notice "?" in lots of columns, we handled that.
* In this step we noticed the ration of categories in categorical data, and we can see how a lot of categories are very dominant and taking over, even the target column "income" isn't balanced.

1. Resetting Data Indexes

Before Encoding, we have to reset indices, because when you filter or manipulate the data frame (e.g., dropping rows of missing values), the index of your data may no longer be sequential, leading to **index mismatches** during concatenation of the encoding process. This causes pandas to fill the mismatched rows with NaN values, **By Resetting the Index**, you ensure that both the original Data Frame and the newly encoded columns have **matching indices**. This prevents pandas from creating extra rows filled with NaN values when concatenating the data.

1. Encoding

A combination of encoding criteria

- encoded the columns that has from 2 to 7 categories using one hot encoding, as it works well with linear models and generally a good idea to limit one-hot encoding to columns with a **small to moderate number of categories**.

- other columns i used target encoding, this technique is especially useful for categorical features with a large number of unique categories and it's Suitable for linear models.

1. Handling Outliers

I used box blot and z-score to discover the errors and how many are they on the numerical data, I appended them in a list then dropped them

1. Feature Scaling (Standardization)

As linear models work better with standardization

1. Feature Engineering

Accuracy before any feature engineer F1 score=> logistic: 0.63

Svm: 62.7

* Removing the educational-num column as it's highly correlated with education column, logistic: 0.621
* Removing "education" column => logistic: 0.631
* Combining marital status columns into 1 column (married or not) = > logistic: 0.627
* Removing some columns that have low correlation with the target ['workclass\_State-gov', 'fnlwgt','workclass\_Without-pay'] => logistic: 0.633, SVM: 62.2

I tried to add more but the accuracy was going down.

**Last accuracy logistic: 0.633, SVM: 62.4 was with using**

* Standardization
* z-score outlier handling
* Removing ['workclass\_State-gov', 'fnlwgt','workclass\_Without-pay', education]
* Dropping 'capital-gain', 'capital-loss'
* Combination of encoding

# Iterate and Experiment:

* Only Freeze standardization (no feature scaling), F1-score => logistic = 0.616, SVM = 0.55
* Only Freeze outlier Handling F1-score => 0.623, SVM = 0.621
* Outlier Handling using quartile range => logistic = 0.63, SVM = 0.61
* Feature Scaling (Normalization) => logistic = 0,61, SVM = 0.599
* Not Dropping 'capital-gain', 'capital-loss' => logistic = 0,674, SVM = 0.664
* Combining 'capital-gain', 'capital-loss'

1 - df['net\_capital'] = df['capital-gain'] - df['capital-loss'] => logistic = 0,63, SVM = 0.665

* 2- df['net\_capital'] = df['capital-gain'] \* df['capital-loss'] => logistic = 0,653, SVM = 0.641

**Important Notes Considering Preprocessing:**

Let's analyze the steps again:

**Drop useless columns**

In here it might seem like sparse columns are useless, but this isn't always the case, despite being filled mostly with zeros—may still have contributed to improving the accuracy of your model due to the **information embedded in those zeros** and the significance of the **non-zero values**.

**Encoding**

By doing combination of one-hot and target encoding as we mentioned before, Note, I see it's better to do one-hot encoding BEFORE splitting the data, one hot encoding is suitable if the columns has up to 10 categories, If this is our case, if we applied one hot encoding before splitting and the data was not balanced, you might find that there are classes in the train that doesn't exist in the test (or vice-versa), the problem in that a class is not exist in train will have some encoding in the test will be indeed similar to a classes was already encoded in the train. **so that lead to having a class in test have the same encoding as some class in the train, but it's different from it.**

**Any way this case won't happen in our code because I used one-hot only on columns that have up to 7 categories and all categories exist in both train and test.**

Of course, target encoding should happen after split to avoid data leakage and there for over fitting.

**Handling Outliers**

**Z-Score Method** was higher in accuracy, it tends to be more conservative, **as it only flags extreme outliers** that are many standard deviations away from the mean, This method allows you to **keep most of the data**, especially if your data distribution is approximately normal or if you set the threshold at a higher Z-score (e.g., 3 standard deviations). By doing so, only a few extreme outliers are removed.

**(IQR) Method**

can be more **aggressive**, especially in **skewed data**. It can flag many data points as outliers if the distribution is long-tailed or skewed, even if they aren’t extreme, this is because the IQR method doesn’t take into account the overall shape of the data distribution (such as whether it's skewed or normal), and it can flag moderate outliers that aren’t extreme but still fall outside the whiskers of the box plot, **it have too many data points, including valuable data points that were not harmful outliers, leading to a decrease in accuracy**.

**Feature Scaling**

Standardization is important and it's proved that is' suitable for the data considering some factors like Models that Rely on Distance and Weighting **(linear models)** which perform better with **standardized data** because they assume that all features contribute equally to the outcome.

The dataset also contains features with **different ranges and units** (capital-gain, age), where some features have larger variances and others are more tightly distributed. Standardization helps scale all features to the same variance, ensuring that **features with large ranges** (like capital-gain) don’t dominate the model over features with smaller ranges (like age).

And it's also less sensitive to outliers unlike normalization etc.

**Feature Engineering**

**Feature removal**

Many features might seem to have less correlation with the target, when I tried to removed it the accuracy decreased, **Low Correlation Doesn't Mean No Contribution**, **Correlation** measures the **linear relationship** between two variables, and the features may contribute to the model in a **non-linear** way, it even might interact with other features in ways that aren't captured by simple correlation.

**Feature combining**

**We Combine features when there’s a meaningful relationship between them,** combining have a lot of benefits **like two features may interact in a way that influences the target variable, and their combined effect might be stronger or more meaningful,** Reduce Dimensionality or even If the relationship between features and the target is non-linear, combining features can help models to better fit the data, **but** **also** Combining two features into one can sometimes reduce accuracy because the new combined feature may result in a **loss of information** or **distortion** of the original relationships between the features and the target variable **like in our case.**

It a trade-of, you might risk losing some accuracy in order to Reduce Dimensionality for instance.