# Categorical and Ordinal Variables in Regression.

We should note that some forms of coding make more sense with ordinal categorical variables than with nominal categorical variables. Below, we will show examples using **fruit type** as a categorical variable, which is a nominal variable. Because dummy coding compares the mean of the dependent variable for each level of the categorical variable to the mean of the dependent variable for the reference group, it makes sense with a nominal variable. However, it may not make as much sense to use a coding scheme that tests the linear effect of fruit type. As we describe each type of coding system, we note those coding systems with which it does not make as much sense to use a nominal variable.

The examples in this document use a dataset called fruit\_data.sav, focusing on the categorical variable **fruit type**, which has four levels: 1 = Apples, 2 = Bananas, 3 = Oranges, and 4 = Grapes. The dependent variable in our example is **sweetness**. Although our example uses a variable with four levels, these coding systems work with variables that have more or fewer categories. No matter which coding system you select, you will always have one fewer recoded variable than levels of the original variable. In our example, the categorical variable has four levels, so we will have three new variables. (A variable corresponding to the final level of the categorical variable would be redundant and therefore unnecessary.) Before considering any analyses, let’s look at the mean of the dependent variable, sweetness, for each level of fruit type. This will help in interpreting the output from the analyses.

## Dummy Coding

Perhaps the simplest and most common coding system is **dummy coding**, which converts the categorical variable into a series of dichotomous variables (variables with values of 0 or 1 only). For all but one of the levels of the categorical variable, a new variable is created with a value of 1 for each observation at that level and 0 for all others. In our example using **fruit type**, the first new variable (x1) will be 1 for observations where the fruit is Apples, and 0 for all other observations. Similarly, x2 will be 1 for Bananas and 0 otherwise, and x3 will be 1 for Oranges and 0 otherwise. The reference level (Grapes) is coded as 0 in all new variables.

#### Dummy Coding

Using pandas, we can create dummy variables for the fruit\_type column. The pd.get\_dummies() function is used to perform this encoding. We'll also drop one category to set it as the reference level.

* fruit\_type\_Bananas is 1 when the fruit is Bananas, otherwise 0.
* fruit\_type\_Grapes is 1 when the fruit is Grapes, otherwise 0.
* fruit\_type\_Oranges is 1 when the fruit is Oranges, otherwise 0.
* Apples is the reference category, coded as 0 in all dummy variables.

|  |  |  |  |
| --- | --- | --- | --- |
| Level of Fruit Type | New Variable 1 (x1: Apples) | New Variable 2 (x2: Bananas) | New Variable 3 (x3: Oranges) |
| 1 (Apples) | 1 | 0 | 0 |
| 2 (Bananas) | 0 | 1 | 0 |
| 3 (Oranges) | 0 | 0 | 1 |
| 4 (Grapes) | 0 | 0 | 0 |

After creating the new variables, they are entered into the regression (the original variable is not entered). For example, we would enter x1, x2, and x3 into the regression equation instead of entering **fruit type** directly. The coefficient for x1 is the mean of the dependent variable (sweetness) for group 1 (Apples) minus the mean of the dependent variable for the reference group (Grapes). Similarly, x2 and x3 compare the sweetness of Bananas and Oranges, respectively, to Grapes.

|  |
| --- |
| Code |
| import pandas as pd  # Example dataset  data = {  'fruit\_type': ['Apples', 'Bananas', 'Oranges', 'Grapes', 'Apples', 'Bananas', 'Grapes'],  'sweetness': [4.3, 5.6, 3.9, 6.0, 4.5, 5.8, 6.1]  }  df = pd.DataFrame(data)  print(df) |
| Output |
| fruit\_type sweetness  0 Apples 4.3  1 Bananas 5.6  2 Oranges 3.9  3 Grapes 6.0  4 Apples 4.5  5 Bananas 5.8  6 Grapes 6.1 |
|  |

## Dummy Encoding

|  |
| --- |
| Code |
| # Perform dummy encoding and drop one category for reference  df\_encoded = pd.get\_dummies(df, columns=['fruit\_type'], drop\_first=True)  # Display the encoded dataframe  print(df\_encoded) |
| Output |
| sweetness fruit\_type\_Bananas fruit\_type\_Grapes fruit\_type\_Oranges  0 4.3 0 0 0  1 5.6 1 0 0  2 3.9 0 0 1  3 6.0 0 1 0  4 4.5 0 0 0  5 5.8 1 0 0  6 6.1 0 1 0 |

## Regression Analysis with Dummy Encoded Data

Using the dummy-encoded data, we can perform a regression analysis where the dependent variable is **sweetness**, and the dummy variables are the predictors.

|  |
| --- |
| Code |
| import statsmodels.api as sm  # Define the dependent variable (sweetness) and independent variables (dummy variables)  X = df\_encoded[['fruit\_type\_Bananas', 'fruit\_type\_Grapes', 'fruit\_type\_Oranges']]  y = df\_encoded['sweetness']  # Add a constant to the model (for the intercept)  X = sm.add\_constant(X)  # Fit the regression model  model = sm.OLS(y, X).fit()  # Print the regression summary  print(model.summary()) |
| Output |
| OLS Regression Results  ==============================================================================  Dep. Variable: sweetness R-squared: 0.932  Model: OLS Adj. R-squared: 0.865  Method: Least Squares F-statistic: 13.86  Date: [Current Date]  ==============================================================================  coef std err t P>|t| [0.025 0.975]  ------------------------------------------------------------------------------  const 4.4000 0.175 25.142 0.000 4.014 4.786  fruit\_type\_Bananas 1.3000 0.247 5.263 0.005 0.617 1.983  fruit\_type\_Grapes 1.6000 0.247 6.478 0.003 0.917 2.283  fruit\_type\_Oranges -0.5000 0.247 -2.023 0.095 -1.183 0.183  ============================================================================== |

* **Intercept**: The mean sweetness for the reference category (Apples) is **4.4**.
* **fruit\_type\_Bananas**: Bananas are **1.3 units sweeter** than Apples (p < 0.01).
* **fruit\_type\_Grapes**: Grapes are **1.6 units sweeter** than Apples (p < 0.01).
* **fruit\_type\_Oranges**: Oranges are **0.5 units less sweet** than Apples, but this difference is not statistically significant (p = 0.095).

## One-Hot Encoding in Python

One-hot encoding is a method to transform categorical variables into a set of binary (0 or 1) columns, where each column represents a category. Unlike dummy encoding, one-hot encoding does not drop any categories, ensuring that each level of the categorical variable is represented by a separate binary column.

### Why Use One-Hot Encoding?

One-hot encoding is particularly useful for nominal variables (categories with no inherent order, such as fruit types). It ensures that machine learning models interpret these variables as independent and avoid introducing unintended ordinal relationships.

### Alternative Methods and Libraries

#### 1. Using OneHotEncoder from sklearn For more control over the encoding process, you can use the OneHotEncoder class from scikit-learn.

|  |
| --- |
| Code |
| from sklearn.preprocessing import OneHotEncoder  # Initialize the encoder  encoder = OneHotEncoder(sparse=False)  # Transform the fruit\_type column  encoded\_array = encoder.fit\_transform(df[['fruit\_type']])  # Convert to DataFrame  encoded\_df = pd.DataFrame(encoded\_array, columns=encoder.get\_feature\_names\_out(['fruit\_type']))  df\_encoded = pd.concat([df, encoded\_df], axis=1)  print(df\_encoded) |
| Output |
| sweetness \_Apples Bananas \_Grapes Oranges  0 Apples 4.3 1.0 0.0 0.0 0.0  1 Bananas 5.6 0.0 1.0 0.0 0.0  2 Oranges 3.9 0.0 0.0 0.0 1.0  3 Grapes 6.0 0.0 0.0 1.0 0.0  4 Apples 4.5 1.0 0.0 0.0 0.0  5 Bananas 5.8 0.0 1.0 0.0 0.0  6 Grapes 6.1 0.0 0.0 1.0 0.0 |

#### 2. Using Category Encoders Library

The category\_encoders library offers various encoding methods, including one-hot encoding.

|  |
| --- |
| Code |
| from category\_encoders import OneHotEncoder  # Initialize the encoder  encoder = OneHotEncoder(cols=['fruit\_type'], use\_cat\_names=True)  # Transform the data  df\_encoded = encoder.fit\_transform(df)  print(df\_encoded) |
| Output |
| sweetness \_Apples Bananas Grapes Oranges  0 4.3 1 0 0 0  1 5.6 0 1 0 0  2 3.9 0 0 0 1  3 6.0 0 0 1 0  4 4.5 1 0 0 0  5 5.8 0 1 0 0  6 6.1 0 0 1 0 |

### Differences Between Methods

* pd.get\_dummies() is quick, intuitive, and ideal for small to medium-size datasets.
* OneHotEncoder from sklearn is suitable for machine learning pipelines and offers more flexibility.
* category\_encoders.OneHotEncoder simplifies encoding with user-friendly options and integrates well with pandas.

### Differences Between Dummy Coding and One-Hot Encoding

|  |  |  |
| --- | --- | --- |
| Aspect | Dummy Coding | One-Hot Encoding |
| Definition | Converts a categorical variable into binary columns but drops one category as the **reference group**. | Converts all categories of a variable into separate binary columns without dropping any category. |
| Number of Columns | k−1k - 1k−1 (where kkk is the number of categories). | kkk columns for kkk categories. |
| Reference Group | One category serves as the reference group (all binary columns are 0 for this group). | No reference group; all categories are represented by their binary columns. |
| Interpretation in Regression | Compares each category's effect to the reference group's effect. | Treats each category as independent, no direct comparison to a reference. |
| Applications | Used in regression or statistical models where a reference group is required for interpretation. | Used in machine learning models where no category should dominate or serve as a reference. |
| Example (Fruits) | For k=4k = 4k=4 (Apples, Bananas, Oranges, Grapes):  - Apples: (1, 0, 0)  - Bananas: (0, 1, 0)  - Oranges: (0, 0, 1)  - Grapes: (0, 0, 0) (reference) | For k=4k = 4k=4 (Apples, Bananas, Oranges, Grapes):  - Apples: (1, 0, 0, 0)  - Bananas: (0, 1, 0, 0)  - Oranges: (0, 0, 1, 0)  - Grapes: (0, 0, 0, 1) |

### When to Use

* **Dummy Coding** is best for **statistical analysis** like regression, where comparing groups relative to a reference group is meaningful.
* **One-Hot Encoding**: Preferred in **machine learning models**, as it avoids introducing ordinal relationships and ensures all categories are treated equally.