Pre-processing & Training Data

Overview of Preprocessing Workflow

The preprocessing stage in this project is designed to refine the dataset to enhance the performance of predictive models. The data_transformation.py script, building on the initial data setup from data_ingestion.py, tackles tasks such as handling class imbalances, imputing missing values, encoding categorical variables, and scaling features. This script accesses data from intermediate pickle files, specifically working with dataset number 24 out of 26 datasets processed to date. It represents the 25th iteration in a series of 29 experiments, each designed to optimize different aspects of the preprocessing workflow.

Feature Selection and Splitting

To focus on variables that significantly affect the target variable 'isMissionSuccess', irrelevant features are removed from the dataset. The remaining data is split into training and testing sets, with a test size set at 5%. This proportion is carefully chosen to maximize the volume of data available for training while providing sufficient data for reliable model validation, ensuring robust model performance in real-world applications.

Handling Data Imbalance

In the context of predicting mission statuses for space launches, the imbalance is stark. As illustrated in Figure 1, successful missions dominate the dataset, comprising 89.1% of the total. The minority classes—failures, partial failures, and prelaunch failures—make up just 7.8%, 2.3%, and 0.1%, respectively. The primary challenge with such an imbalance is that predictive models might become overly trained to recognize and predict the majority class, leading to a high overall accuracy but poor predictive performance for the minority class. This situation is particularly problematic when the consequences of failing to predict a minority class (like a mission failure) are significant.

Given the dataset contains just over 4,000 records, under sampling the majority class would further reduce the dataset size, which is not ideal for maintaining a model's ability to generalize well. Therefore, oversampling the minority class is the preferred approach. This method involves artificially increasing the size of the minority class by replicating its instances. Oversampling helps achieve a balanced class distribution without losing valuable data, which is especially important in smaller datasets.

Simplifying the classification from multi-class to binary (grouping all non-success outcomes into a single 'Failure' category) streamlines the problem. This adjustment reduces the complexity that the model needs to handle, enabling it to focus on distinguishing between success and failure effectively.

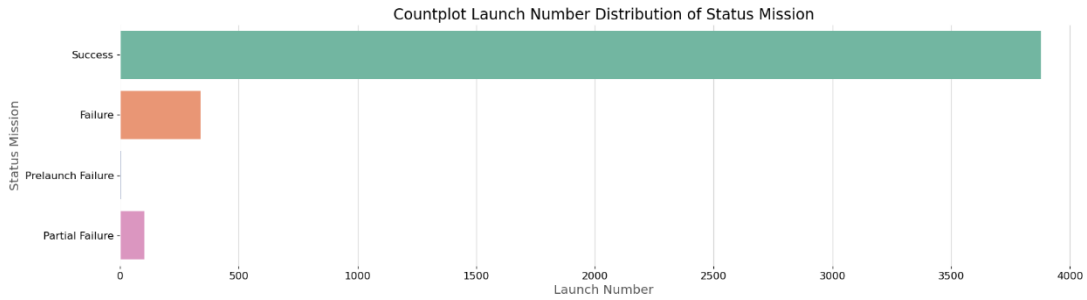

Figure 1

Data Transformation Strategy

The preprocessing phase aims to appropriately handle each feature based on its characteristics, such as data type and distribution. Here's a closer examination of the key features and the rationale for their specific preprocessing approaches. For further detail on each of these figures, refer to the EDA reports:

1. 'Rocket Cost':
    - Data Type and Distribution: This feature is a continuous numerical variable with a wide range from a minimum of $5.3 million to a maximum of $5 billion, indicative of the varied costs associated with different rocket launches. For visual details on this range, see Figure 2.
    - Imputation Strategy: Given the skewed nature of this distribution, median imputation is utilized to handle missing values, with a median value of $59 million. This approach was chosen to avoid the influence of extreme values on the imputation process, ensuring a more robust and representative fill value for missing data.

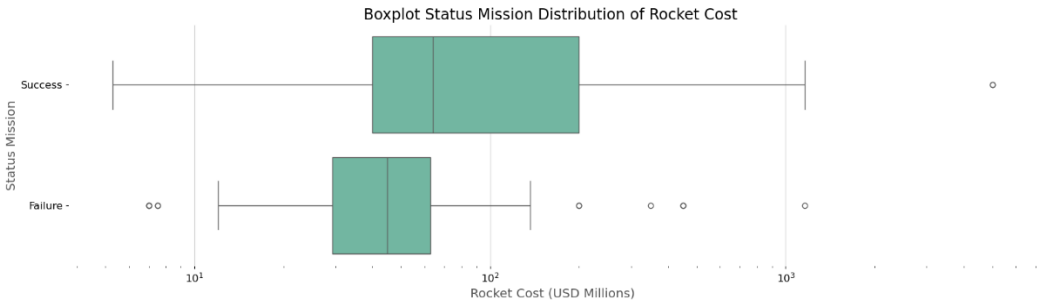

Figure 2

2. 'Year', 'Month', 'Day':
    - Data Type: These are discrete numerical features. 'Year' ranges from 1957 to 2020, 'Month' from 1 to 12, and 'Day' from 1 to 31. For visual details on this range, see Figure 3, 4, and 5.
    - Scaling: StandardScaler is applied to these features to normalize their scale. Although these are inherently discrete, their numerical nature and range make them suitable for scaling to ensure they contribute equally when modeling, avoiding any potential scale bias that could affect the algorithm's performance.

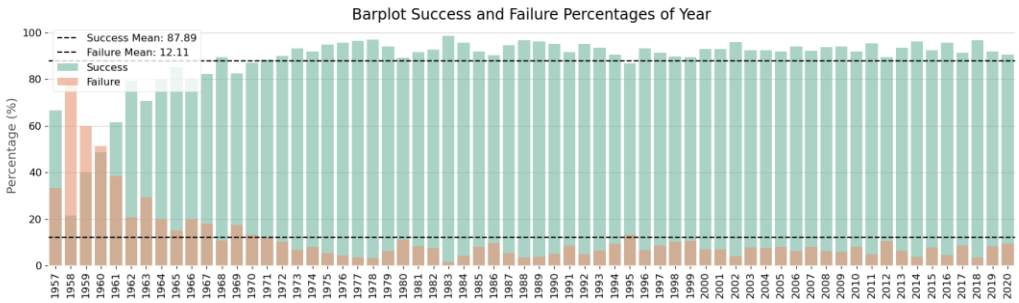

Figure 3

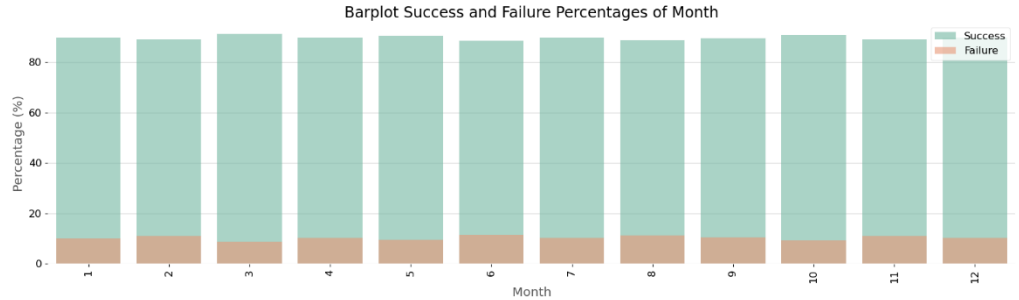

Figure 4

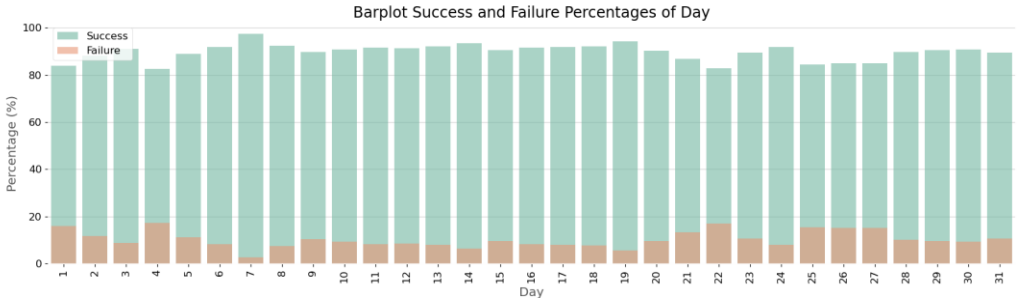

Figure 5

3. Geographical Coordinates ('Company Origin Lat', 'Company Origin Long'):
    - Data Type: Continuous numerical variables representing latitude and longitude. Figure 6 presents scatterplots of the geographical distribution for company origins, which reveal major clusters of launch activities in the US and Russia, predominantly situated in the northern and eastern hemispheres. See Figure 7 for a global map highlighting these hemispheres.
    - Scaling: Given their float data type and range from negative to positive values, scaling these features is crucial. StandardScaler is used to ensure these geographically based measurements do not overpower other variables in model calculations due to their range and scale.

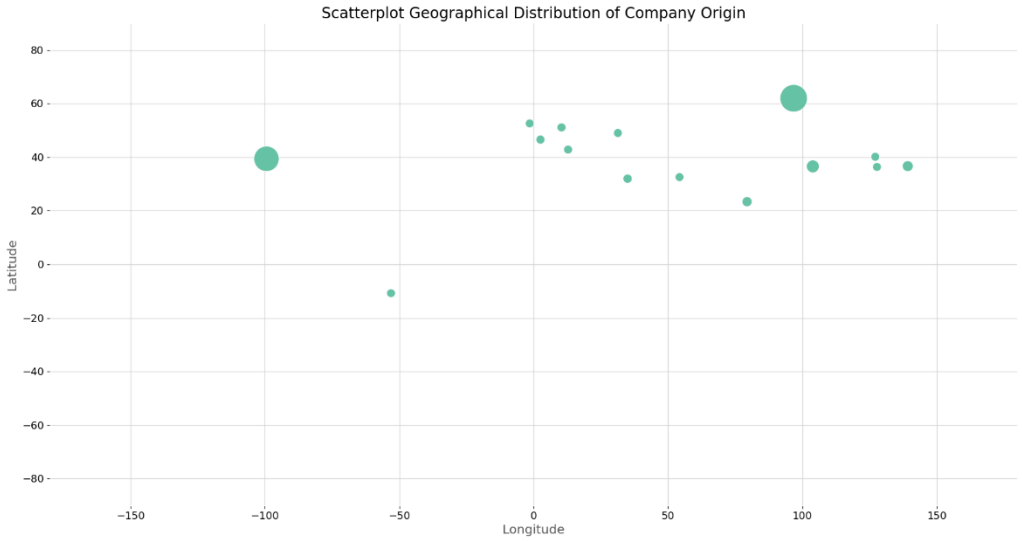

Figure 6

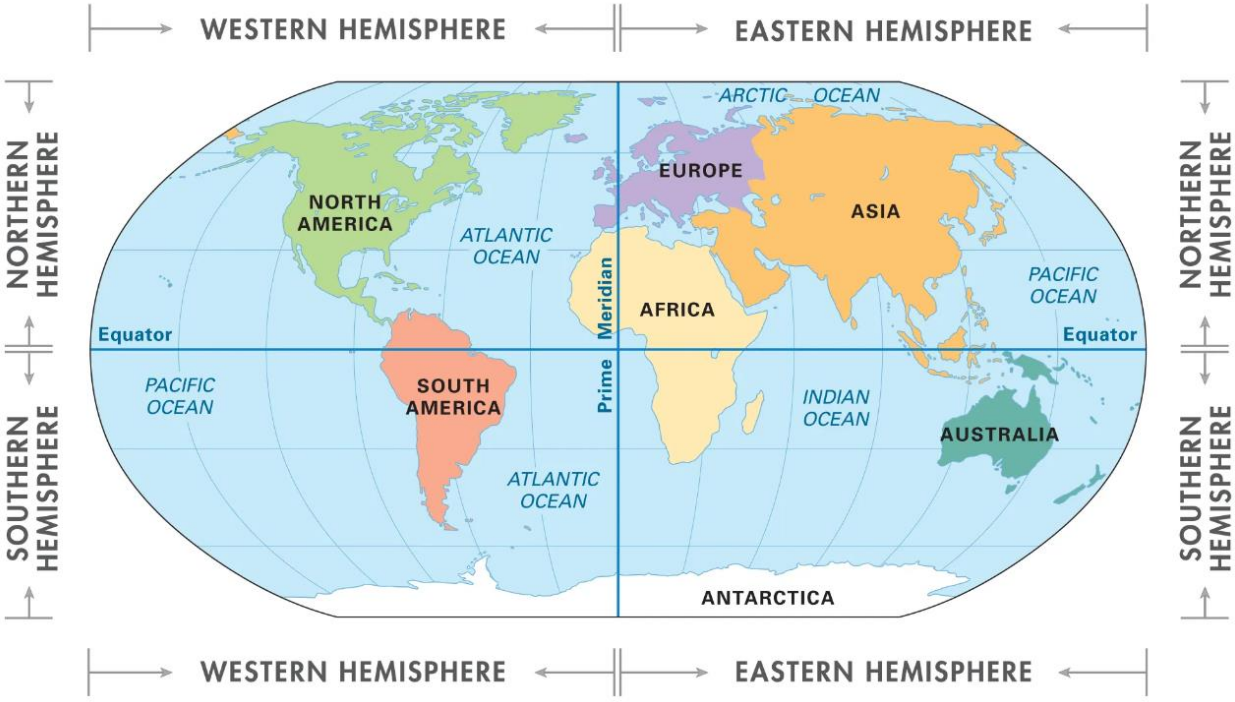

Figure 7

4. 'Unix Time':
    - Data Type: Continuous numerical variable, ranging from -0.4e9 to 1.6e9. For visual details on this range, see Figure 8.
    - Scaling: The significant range of Unix Time values, which represent timestamps, benefits from scaling to normalize these values within a comparable range to other features, enhancing model accuracy and learning efficiency.

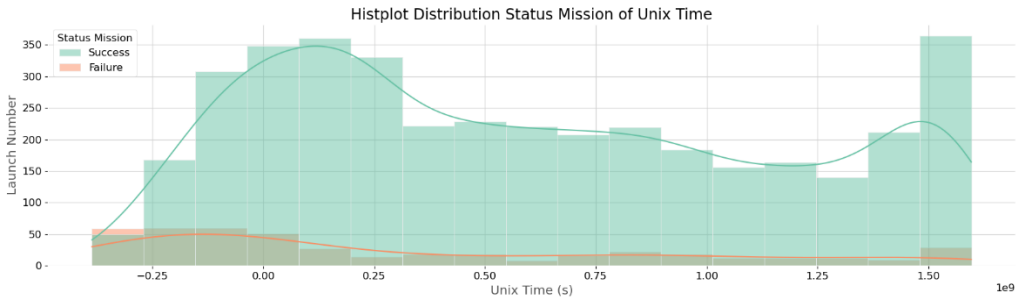

Figure 8

5. 'YearSine':
    - Data Type: Continuous numerical variable, ranging from -1 to 1. For visual details on this range, see Figure 9.
    - Scaling: Although already within a standard range, applying StandardScaler helps maintain consistency in preprocessing across all continuous numerical variables, ensuring that this cyclic feature integrates well with other features in the model.

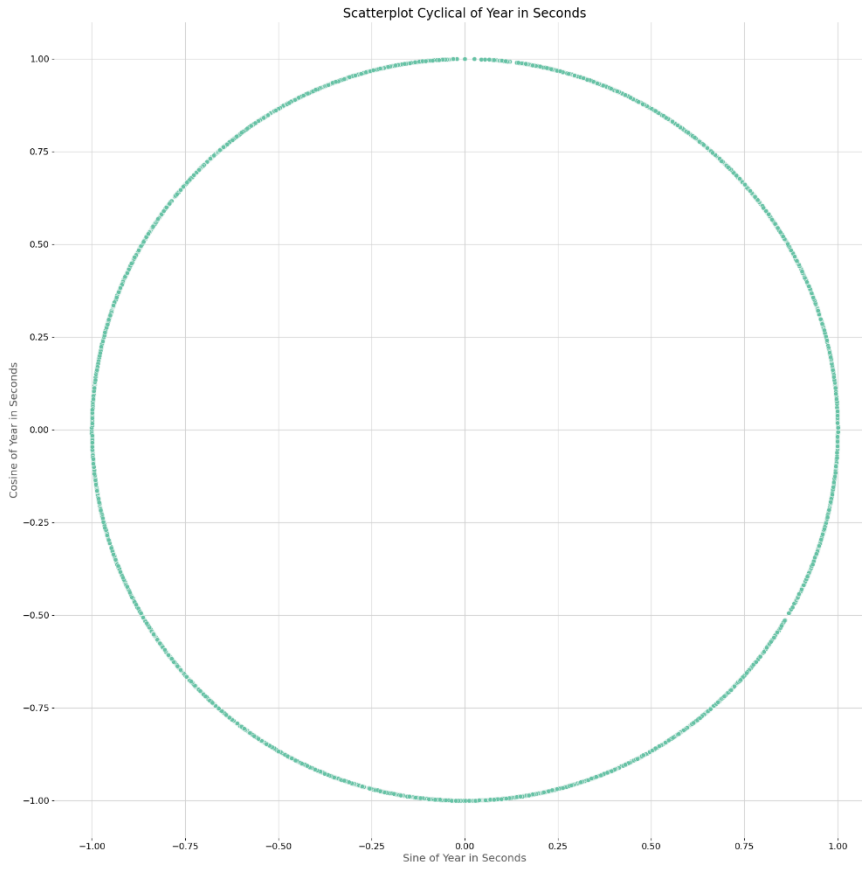

Figure 9

6. Categorical Variables ('Status Rocket', 'isWeekend'):
    - Data Type and Categories: 'Status Rocket' has categories like StatusActive and StatusRetired; 'isWeekend' is a binary feature with values True or False. For visual details on these categories, see Figure 10 and 11.
    - Encoding: These features are transformed using OrdinalEncoder to convert them into a numerical format suitable for machine learning models. This encoding is straightforward as the features are binary or have a clear order, making them ideal for ordinal encoding.

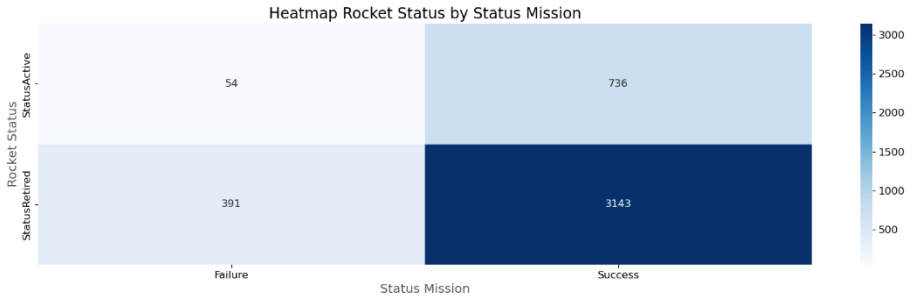

Figure 10

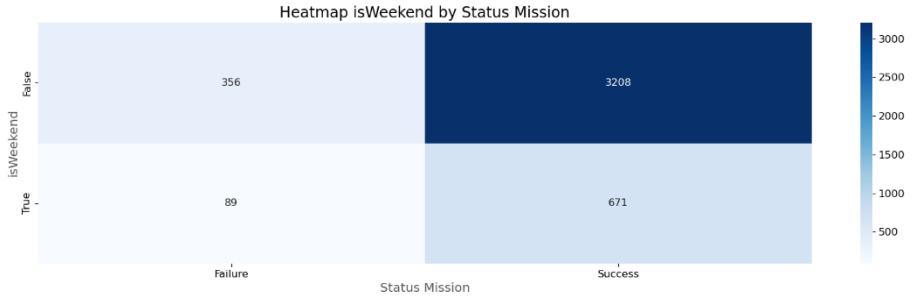

Figure 11

Exclusion of OneHotEncoder Features:

- No features that required OneHotEncoder proved useful in enhancing the model's performance metrics, leading to their exclusion from this experiment. This decision highlights the importance of feature selection in optimizing model outcomes, ensuring only impactful features are included in the final model.