Function for automating feature selection

```
import pandas as pd

import numpy as np


def prepare_data(data):
    """
    This function prepares the data for training by separating the features and target variable.
    It assumes that the target variable is always the last column in the pandas DataFrame,
    and the features are all the columns up to that.
    The function accounts for missing values by imputing the mean for numerical columns and the mode for categorical columns.
    It additionally checks if the input data is a dictionary, DataFrame, or a file path (CSV/Excel) and processes it accordingly,
    attempting to load the data if it's a file path. In the case of a dictionary, it converts it to a DataFrame.
    If the dataframe is empty, has only one column, or has no target column, the function raises a ValueError.


    Parameters:
    data (pd.DataFrame, dict, str): The dataset to prepare. It can be a DataFrame, dictionary, or a file path (CSV/Excel).


    Returns:
    X (np.array): The features
    y (np.array): The target variable
    """
    if isinstance(data, dict):
        data = pd.DataFrame(data)
    elif isinstance(data, str):
        if data.endswith('.csv'):
            data = pd.read_csv(data)
        elif data.endswith('.xlsx') or data.endswith('.xls'):
            data = pd.read_excel(data)
```

```
        else:
             raise ValueError("Unsupported file format. Please provide a CSV or Excel file.")
    elif not isinstance(data, pd.DataFrame):
        raise ValueError("Input data must be a pandas DataFrame, dictionary, or a file path (CSV/Excel).")

    if data.empty:
        raise ValueError("The input DataFrame is empty.")

    if data.shape[1] < 2:
          raise ValueError("The input DataFrame must have at least one feature column and one target column.")

    for column in data.columns:
        if data[column].dtype == np.number:
            data[column].fillna(data[column].mean(), inplace=True)
        else:
            data[column].fillna(data[column].mode()[0], inplace=True)

    feature_columns = data.columns[:-1]
    target_column = data.columns[-1]


    X = data[feature_columns].values
    y = data[target_column].values

    return X, y
```

Function for computing accuracy of the predictions made by the stochastic gradient

```
def calculate_accuracy(y_true, y_pred):
    """
    This function calculates and returns various accuracy metrics for the model.
```

```
    Parameters:
    y_true (np.array): The true target values
    y_pred (np.array): The predicted target values

    Returns:
    dict: A dictionary containing various accuracy metrics

    Metrics:
    - Mean Absolute Error (MAE): The average of the absolute differences between the predicted and actual values.
    - Mean Squared Error (MSE): The average of the squared differences between the predicted and actual values.
    - Root Mean Squared Error (RMSE): The square root of the average of the squared differences between the predicted and actual values.
    - R-squared (R²): The proportion of the variance in the dependent variable that is predictable from the independent variables.
    - Mean Absolute Percentage Error (MAPE): The average of the absolute percentage differences between the predicted and actual values.
    - Explained Variance Score (EVS): Measures the proportion of the variance in the target variable that is explained by the model.
    """
    import numpy as np

    metrics = {}
    metrics['MAE'] = np.mean(np.abs(y_true - y_pred))

    metrics['MSE'] = np.mean((y_true - y_pred) ** 2)

    metrics['RMSE'] = np.sqrt(metrics['MSE'])

    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    metrics['R2'] = 1 - (ss_residual / ss_total)
```

```
metrics['MAPE'] = np.mean(np.abs((y_true - y_pred) / y_true)) * 100


variance_y_true = np.var(y_true)

variance_residual = np.var(y_true - y_pred)

metrics['EVS'] = 1 - (variance_residual / variance_y_true)


return metrics
```