AutoProcess: Automated Data Preprocessing Library

Utilize AI-driven preprocessing to build faster, smarter, and context-aware data pipelines

What is AutoProcess?

- A Python library that automates data preprocessing tasks using Google's Gemini AI.

- Focused on generating production-ready code for:

- Data Cleaning
- Feature Engineering
- Skew Correction
- Data Transformation

Why AutoProcess:

- Automates repetitive preprocessing tasks, saving time and effort.
- Ensures consistency and high-quality data processing.
- Minimizes manual coding errors for improved reliability.
- Adapts to specific problems with context-aware preprocessing.

```
pip install autoprocess_iitg
```

```
pip show autoprocess_iitg
```

```
Name: autoprocess_iitg
Version: 0.1.1
Summary: Automated data preprocessing library using Google's Gemini AI, a small change in skew function
Home-page: https://github.com/ShubhamS1101/CleanGPT01
Author: Shubham
Author-email: shubhamsinghalswm123@gmail.com
License: MIT
Location: /usr/local/lib/python3.10/dist-packages
Requires: numpy, pandas, scikit-learn
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

```
from autoprocess import *
```

```
pipeline = FeatureEngineeringPipeline('AIzaSyAxWes9R9o1Gjy_3z4UaAp8OLYUoE8ketI')
pipeline2 =DataCleaningPipeline('AIzaSyAxWes9R9o1Gjy_3z4UaAp8OLYUoE8ketI')
pipeline3 =DataTransformationPipeline('AIzaSyAxWes9R9o1Gjy_3z4UaAp8OLYUoE8ketI')
pipeline4= SkewCorrectionPipeline('AIzaSyAxWes9R9o1Gjy_3z4UaAp8OLYUoE8ketI')
```

Pipeline

- **Dataset Analysis** : The pipeline first generates a detailed description of the dataset.

- **Strategy Generation** : This dataset description is passed to a large language model (LLM) to generate an appropriate preprocessing strategy.

- **Code Generation** : The strategy is then fed into the LLM to generate executable preprocessing code.

- **Iterative Refinement** : The generated code undergoes multiple iterations, each time being passed back to the LLM for optimization and refinement.

Dataset description

- A **helper function** that generates a **concise dataset description**.
- Called **every time** to ensure up-to-date dataset insights.
- Helps in creating **context-aware preprocessing strategies**.

```python
df = pd.read_csv(r"C:\Users\91701\Downloads\train.csv")
```
✓ 0.0s

```python
gen_des(df, sample_size=3)
```
✓ 0.1s


```
{'columns': {'Id': {'dtype': 'int64',
   'missing_pct': 0.0,
   'unique_count': 1460,
   'example_values': [893, 1106, 414, 523, 1037],
   'min': 1.0,
   'max': 1460.0,
   'mean': 730.5,
   'std': 421.61,
   'skew': 0.0},
  'MSSubClass': {'dtype': 'int64',
   'missing_pct': 0.0,
   'unique_count': 15,
   'example_values': [20, 60, 30, 50, 20],
   'min': 20.0,
   'max': 190.0,
   'mean': 56.9,
   'std': 42.3,
   'skew': 1.41},
  'MSZoning': {'dtype': 'object',
   'missing_pct': 0.0,
   'unique_count': 5,
   'example_values': ['RL', 'RL', 'RM', 'RM', 'RL'],
   'value_distribution': {'top_values': ['RL', 'RM', 'FV'],
    'percentages': [78.8, 14.9, 4.5]}},
```

DataCleaningPipeline

•Handles **null values**, **outliers**, and **duplicates** automatically.

•Users can **customize preprocessing** by disabling specific operations (e.g., outlier=False)

•Ensures flexibility while maintaining data integrity and quality.

```
def data_clean(self, dataset, target: str = "", outlier=True, missing=True, duplicate=True)
```

DataTransformationPipeline

•Supports **encoding of categorical columns**, **datatype handling**, and **scaling/normalization**.

•Users can **customize operations** by enabling or disabling specific steps (e.g., datatype=False).

```
def generate_transformation_code(
    self,
    dataset,
    target: str = "",
    skip_encoding: List[str] = None,
    skip_normalisation: List[str] = None,
    max_iterations: int = 3
) -> Dict[str, Any]:
```

FeatureEngineeringPipeline

- We can pass target column so that it can generate features relevant to that.
- This includes festuring new columns and dropping unnecessary columns for target column .

```
def generate_features(self, dataset, target: str, drop_columns: bool = True, max_iterations: int = 3)
```

SkewCorrectionPipeline

- Includes **unskewing techniques** to normalize the target column.
- Helps improve **data distribution** for better model performance.
- Ensures **robust preprocessing** for skewed datasets.

.

```
def generate_skew_correction(self, dataset, column_name, max_iterations=3):
```

```python
import pandas as pd

df = pd.DataFrame({
    'age': [25, 30, 35, 40],
    'income': [50000, 60000, 70000, 80000],
    'purchase_date': pd.date_range(start='2021-01-01', periods=4, freq='D')
})

result = pipeline3.generate_transformation_code(dataset=df)

if "code" not in result:
    raise Exception("Code generation failed: " + result.get("error", "Unknown error"))

generated_code = result["code"]
print( generated_code)
```

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer

def transform_data(df):
    # Datatype handling
    if pd.api.types.is_datetime64_any_dtype(df['purchase_date']):
        df['purchase_date_transformed'] = pd.to_datetime(df['purchase_date']).astype('int64') // 10**9 # Convert to Unix timestamp

    # Categorical encoding
    categorical_features = ['age', 'income']
    skip_categorical = []
    categorical_features = [col for col in categorical_features if col not in skip_categorical]
    ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), categorical_features)], remainder='passthrough')
    encoded_data = ct.fit_transform(df)
    encoded_df = pd.DataFrame(encoded_data, columns=ct.get_feature_names_out())
    df = df.join(encoded_df)

    # Scaling/Normalization
    numerical_features = ['age', 'income']
    skip_numerical = ['purchase_date_transformed']
    numerical_features = [col for col in numerical_features if col not in skip_numerical]
    scaler = MinMaxScaler()
    for col in numerical_features:
        df[f'{col}_transformed'] = scaler.fit_transform(df[[col]])

    return df

```

Thank you