Documentation :

Importing Libraries

```
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import mean_squared_error, r2_score
```

This section imports the necessary libraries for data manipulation (json, pandas, numpy), machine learning model training (sklearn), and evaluation.

Loading JSON File with Error Handling

```
try:
    with open('algoparams_from_ui.json') as f:
        params = json.load(f)
except FileNotFoundError:
    raise FileNotFoundError("The JSON file was not found.")
except json.JSONDecodeError:
    raise ValueError("Error decoding JSON file. Please check the file format.")
```

This block attempts to load a JSON file containing model parameters and handles errors if the file is not found or if there's a problem with JSON formatting.

Loading CSV File with Error Handling

```
try:
    data = pd.read_csv('iris.csv')
except FileNotFoundError:
    raise FileNotFoundError("The CSV file was not found.")
except pd.errors.ParserError:
    raise ValueError("Error parsing CSV file. Please check the file format.")
```

This block loads the dataset from a CSV file and handles errors if the file is not found or if there is an issue with parsing.

Validate the JSON Structure

```
required_keys = ['design_state_data']
if not all(key in params for key in required_keys):
    raise KeyError("Missing required keys in JSON.")

design_data = params['design_state_data']
```

This section checks that the JSON contains the required keys and extracts the design data.

Validate Target and Prediction Type

```
if 'target' not in design_data or 'prediction_type' not in design_data['target']:
    raise KeyError("Missing target or prediction type information in JSON.")

target = design_data['target']['target']
prediction_type = design_data['target']['prediction_type']
```

This part validates the presence of target and prediction type information in the JSON.

Validate and Extract Features

```
if 'feature_handling' not in design_data:
    raise KeyError("Missing feature handling information in JSON.")

features = [
    feature for feature in design_data['feature_handling'].keys()
    if design_data['feature_handling'][feature]['is_selected']
]

if not features:
    raise ValueError("No features selected. Please select at least one feature.")
```

This section checks for the feature handling information and extracts the selected features.

Prepare the Data

```
X = data[features]
y = data[target]

# Encode the target variable if it's categorical
if y.dtype == 'object':
    le = LabelEncoder()
    y = le.fit_transform(y)

# Identify categorical and numerical features
categorical_features = X.select_dtypes(include=['object']).columns.tolist()
numerical_features = X.select_dtypes(include=[np.number]).columns.tolist()
```

This block prepares the data by separating features (X) and target (y), encoding the target if it's categorical, and identifying categorical and numerical features.

Impute Missing Values with Validation

```
imputer_strategies = {}
for feature in features:
    if feature in design_data['feature_handling']:
        feature_details = design_data['feature_handling'][feature]['feature_details']
        if 'missing_values' in feature_details and feature_details['missing_values'] == 'Impute':
            strategy = 'mean' if feature_details['impute_with'] == 'Average of values' else 'constant'
            fill_value = feature_details.get('impute_value', 0)
            imputer_strategies[feature] = (strategy, fill_value)
```

This part identifies imputation strategies for missing values based on the JSON configuration.

Setup Column Transformer for Preprocessing

```
transformers = []

for feature in numerical_features:
    if feature in imputer_strategies:
        strategy, fill_value = imputer_strategies[feature]
        if strategy == 'mean':
            transformers.append((feature, SimpleImputer(strategy='mean'), [feature]))
        else:
            transformers.append((feature, SimpleImputer(strategy='constant', fill_value=fill_value),
[feature]))
    else:
        transformers.append((feature, SimpleImputer(strategy='mean'), [feature]))

for feature in categorical_features:
    transformers.append((feature, OneHotEncoder(handle_unknown='ignore'), [feature]))

preprocessor = ColumnTransformer(transformers, remainder='passthrough')
```

This block sets up transformers for numerical and categorical features, including imputation and encoding.

Feature Reduction with Validation

```
if 'feature_reduction' not in design_data:
    raise KeyError("Missing feature reduction information in JSON.")

feature_reduction_method = design_data['feature_reduction'].get('feature_reduction_method', 'No
Reduction')

if feature_reduction_method == 'PCA':
    n_components = int(design_data['feature_reduction'].get('num_of_features_to_keep', 2))
    feature_reduction = PCA(n_components=n_components)
elif feature_reduction_method == 'Tree-based':
    tree_model = DecisionTreeRegressor(max_depth=5)
    feature_reduction = SelectFromModel(tree_model)
elif feature_reduction_method == 'Corr with Target':
    corr_threshold = float(design_data['feature_reduction'].get('correlation_threshold', 0.1))
    corrs = X.corrwith(pd.Series(y))
    selected_features = corrs[abs(corrs) > corr_threshold].index.tolist()
    feature_reduction = ColumnTransformer([(col, 'passthrough', [col]) for col in selected_features],
remainder='drop')
else:  # No Reduction
    feature_reduction = 'passthrough'
```

This section configures feature reduction based on the method specified in the JSON (PCA, Tree-based, Correlation with Target, or No Reduction).

Model Selection with Validation

```
if prediction_type == 'Regression':
    if 'algorithms' not in design_data:
        raise KeyError("Missing algorithms information in JSON.")
    if design_data['algorithms']['RandomForestRegressor']['is_selected']:
        model = RandomForestRegressor(
            n_estimators=design_data['algorithms']['RandomForestRegressor'].get('max_trees', 100),
            max_depth=design_data['algorithms']['RandomForestRegressor'].get('max_depth', None)
        )
    else:
        raise ValueError("No valid regression algorithm selected.")
else:
    raise ValueError("Unsupported prediction type. Only 'Regression' is supported.")
```

This block selects the regression model (RandomForestRegressor) based on the JSON configuration.

Setup Pipeline

```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('feature_reduction', feature_reduction),
    ('model', model)
])
```

This section creates a pipeline that combines preprocessing, feature reduction, and the model.

Hyperparameter Tuning with GridSearchCV

```
param_grid = {
    'model__n_estimators': [10, 20, 30],
    'model__max_depth': [10, 20, 30]
}
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2')
```

This block sets up GridSearchCV for hyperparameter tuning with a specified parameter grid.

Split the Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This part splits the data into training and testing sets.

Fit the Model with Error Handling

```
try:
    grid_search.fit(X_train, y_train)
except ValueError as e:
    raise ValueError(f"Error during model fitting: {e}")
```

This block fits the model to the training data and handles any errors that occur during fitting.

Predict and Evaluate Metrics

```
y_pred = grid_search.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
```

This section makes predictions on the test data and evaluates the model's performance using Mean Squared Error (MSE) and R-squared (R^2) score, then prints the results.

This documentation provides a detailed overview of each step and its purpose, helping to understand the flow and functionality of the code.

Summary:

This code performs a machine learning pipeline setup and execution for a regression task. Below is a summary of the key steps involved:

1. **Import Libraries**:
    - Various libraries are imported, including json, pandas, numpy, and several modules from sklearn.
2. **Load JSON and CSV Files with Error Handling**:
    - The JSON file (algoparams_from_ui.json) containing model parameters and design data is loaded. If the file is not found or the JSON is invalid, an error is raised.
    - The CSV file (iris.csv) containing the dataset is loaded. Errors are raised if the file is not found or if there is an issue parsing the CSV.
3. **Validate JSON Structure**:
    - The JSON structure is validated to ensure required keys are present (design_state_data and nested keys).
4. **Extract Target and Features**:
    - The target variable and prediction type are extracted from the JSON.
    - Features are extracted based on the selection criteria specified in the JSON.
5. **Prepare Data**:
    - Data is prepared by separating features (X) and target (y).
    - Categorical target variables are encoded if necessary.
    - Categorical and numerical features are identified.
6. **Impute Missing Values**:
    - Missing value imputation strategies are identified from the JSON and applied to the respective features.
7. **Setup Column Transformer for Preprocessing**:
    - Transformers for numerical and categorical features are set up, including imputation and encoding.
8. **Feature Reduction**:
    - Feature reduction techniques (PCA, Tree-based, Correlation with Target) are applied based on the JSON specifications.
    - If no reduction is specified, the pipeline passes through all features.
9. **Model Selection**:
    - A model is selected based on the JSON specifications. In this case, RandomForestRegressor is used with specified parameters.
10. **Setup Pipeline**:
    - A pipeline is created combining preprocessing, feature reduction, and the model.
11. **Hyperparameter Tuning with GridSearchCV**:
    - GridSearchCV is used for hyperparameter tuning with a specified parameter grid and cross-validation.
12. **Train-Test Split**:
    - The data is split into training and testing sets.

13. **Fit the Model and Predict**:
    - The model is fitted to the training data, and predictions are made on the test data.
    - Errors during model fitting are handled.
14. **Evaluate Metrics**:
    - The performance of the model is evaluated using Mean Squared Error (MSE) and R-squared (R^2) score.
15. **Output Results**:
    - MSE and R^2 score are printed to summarize the model performance.

This process integrates data preprocessing, feature selection, model selection, hyperparameter tuning, and evaluation into a comprehensive machine learning workflow.