Predicting Car Selling Prices

Machine Learning Tasks

Please run the below cell to install libraries:

```
!pip3 install "pandas==2.2.2" "scikit-learn==1.6.1"
"matplotlib==3.10.0"
```

Please run the below cell to import libraries:

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import boto3
import pandas as pd
from sagemaker import get_execution_role
import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

Task 1: Load and Explore the Dataset

Instructions:

- Import AWS SDK `boto3` and other necessary libraries such as NumPy, Pandas & Sklearn on your notebook.
- Load the dataset from S3 bucket: Build the S3 path for the dataset `car_cleaned_data.csv` using string formatting to concatenate the bucket name, folder name and file key i.e the name of the dataset. **Note**: Bucket name - `car-dataXYZXYZ` (XYZXYZ can be any random integers) & Folder name - `car_cleaned_data`.
- Load and store the dataset using pandas DataFrames.
- Print the first few rows of the dataset to understand its structure and use inbuilt functions to get insights on the dataset.

Hints:

- Sample S3 URI - `s3://bucket_name/folder_name/file_name.csv`

```
### Create S3 path for the dataset
bucket= None
```

```python
data_key = None
data_location = None
###

### Load the dataset
df = None

###

### Analyze the dataset


###
```

Task 2: Feature Engineering

Instructions:

- Create a new feature `car_age` which represents the age of the car in years. Add the feature as a column under the name `car_age` in the dataframe.
- Drop the columns `car_name` and `year` from the dataset.

Hints:

- The car's age can be calculated as `2024 - year`.

```python
### Create new feature: age of the car
data['car_age'] = None

###

### Drop the columns


###
```

Task 3: Define Features and Target Variable

Instructions:

- Define the features `X` by dropping the target variable (selling_price) from the dataset.

- Define the target variable `y` as the selling_price.

```python
### Define the features and target variable
X = None
y = None
###
```

Task 4: Preprocess the Data

Instructions:

- Identify numerical and categorical features present in 'X' and 'y', store it in a list for building a pipeline.

- Apply log transformation to the km_driven and selling_price columns to reduce skewness. Replace the trasformed data back to 'X' & 'y'.
- Use StandardScaler to scale the numerical features.
- Use OneHotEncoder to encode the categorical features, ensuring that unknown categories are handled.

Hints:

- Use `np.log1p()` for log transformation
- Example numerical feature list: `numerical_features = [ ‘xy’, ‘yz’]`
- Example categorical feature list: `categorical_features = [ ‘xy’, ‘yz’, ‘zz’]`

```
### Preprocessing for numerical features & categorical features
numerical_features = None
categorical_features = None


###

### Log transformation for skewed numerical features


###
```

Task 5: Build a Transformer Pipeline

Instructions:

- Create a 'ColumnTransformer' pipeline to preprocess the numerical and categorical features.
- Create a pipeline that includes the preprocessors i.e 'StandardScaler' & 'OneHotEncoder'. Store the transformer pipeline in a variable, for example `preprocessor`. We will utilize this for building the model pipeline in the upcoming steps.

Hints:

- Sample Pipeline:

```
preprocessor = ColumnTransformer(transformers=[
        ('xy', scaler(), feature_list1),
        ('yz', encoder(), feature_list2)
    ])
```

```
### Create the column transformer
preprocessor = ColumnTransformer(

    )

###
```

Task 6: Build a Model Pipeline and Split the Data

Instructions:

- Create a Sklearn model pipeline with the preprocessor pipeline and 'RandomForestRegressor' model with a random state set to `8`. Store it under the name 'model'.
- Split the dataset into training and test sets using 'train_test_split' with a test size of 20% and random state set to '8'.

Hints:

- Sample pipeline:

```
model = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('model', ModelFunc(arg=value))

])
```

```
### Create a pipeline with the preprocessor and the model
model = Pipeline(

)

###

### Split the data into training and test sets


###
```

Task 7: Hyperparameter Tuning

Instructions:

- Perform hyperparameter tuning using GridSearchCV to find the best parameters for the RandomForestRegressor. The parameter grid is provided in the **Hints** below.
- Build the Grid Search named `grid_search` with the generated model pipeline, parameter grid, cross-validation generator set to '5', scoring set to 'r2' and number of jobs set to '1'.

Hints:

- Parameter grid:

```
param_grid = {

    'regressor__n_estimators': [100, 200, 300],

    'regressor__max_depth': [None, 10, 20, 30],

    'regressor__min_samples_split': [2, 5, 10]
}
```

- Define a parameter grid with different values for n_estimators, max_depth, and min_samples_split.

```
### Perform GridSearchCV for hyperparameter tuning
param_grid = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [None, 10, 20, 30],
    'regressor__min_samples_split': [2, 5, 10]
}
###

### Create the model
grid_search = None
###
```

Task 8: Train the Model

Instructions:

- Fit the GridSearchCV on the training data to find the best model. (Note: Fitting process may take a few minutes to complete the execution, please wait until the process is finished.)

- Extract the best model from the grid search results. Store the best model under variable for example `best_model`.

Hints:

- Use `grid_search.best_estimator_` to get the best model.

```
### Best model from grid search



best_model = None

###
```

Task 9: Make Predictions

Instructions:

- Use the best model to make predictions on the test set and store it in variable called `y_pred`.
- Transform the predictions i.e `y_test` and `y_pred` back to the original scale.

Hints:

- Use `np.expm1()` to reverse the log transformation.

```
### Make predictions
y_pred = None

###

### Transform predictions back to original scale
y_test = None
y_pred = None
###
```

Task 10: Evaluate the Model

Instructions:

- Calculate the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score for the model.
- Print the evaluation metrics and the best hyperparameters found.

Hints:

- Use `grid_search.best_params_` to get the best hyperparameter.

```
### Evaluate the model


###

### Print the metrics
print(f'MAE: {None}')
print(f'MSE: {None}')
print(f'RMSE: {None}')
print(f'R2: {None}')
print(f'Best Parameters: {None}')
###
```

Wohooo! □□

You have completed the challenge! Now, please run the below cells □ to save your answers and data for scoring.

```
#### □ Check if the BUCKET_NAME matches your actual S3 bucket name

BUCKET_NAME = bucket

####
```

Now, please run the below cells □

```
#### □ DO NOT MODIFY BLOCK <START> □

variables_to_store = {
    'df': df if 'df' in locals() else None,
    'grid_search': str(type(grid_search)).lower() if 'grid_search' in
locals() else None,
    'y_pred': y_pred if 'y_pred' in locals() else None,
}
#### □ DO NOT MODIFY BLOCK <END> □

#### □ DO NOT MODIFY BLOCK <START> □
import boto3
import joblib
import time
from botocore.exceptions import NoCredentialsError,
PartialCredentialsError
import os
import re

pipe_data = 'model_objects.joblib'

def save_data(filename, data):
    try:
        with open(filename, 'wb') as f:
            joblib.dump(data, f)
        print(f'Data has been saved as: {filename}...')
    except Exception as e:
        print(f'Error saving data to {filename}: {e}')
        return False
    return True

def upload_to_s3(filename):
    try:
        with open(filename, 'rb') as f:
            s3_client.put_object(Body=f.read(), Bucket=BUCKET_NAME,
Key=filename)
```

```
        print(f'Data has been uploaded to S3 as: {filename}...')
        return True
    except FileNotFoundError:
        print(f'File {filename} not found.')
    except NoCredentialsError:
        print('Credentials not available.')
    except PartialCredentialsError:
        print('Incomplete credentials provided.')
    except Exception as e:
        print(f'Error uploading data to S3: {e}')
        print(f'Please check bucket name: {BUCKET_NAME}')
    return False

s3_client = boto3.client('s3')

# Save and upload model objects
if BUCKET_NAME is not None:
    print(f'Step #1 Processing Upload: {pipe_data}...')
    start_time = time.time()
    if save_data(pipe_data, variables_to_store) and
upload_to_s3(pipe_data):
        print(f'Total time taken for Pipeline data: {time.time() -
start_time:.2f} seconds')
        print('□ Step #1 Done!!')
        print('---')
    else:
        print('□ Step #1 failed. Please check the Error!!!')
else:
    print('BUCKET_NAME is set to None or Incorrect. Please assign your
exact bucket name!!')
#### □ DO NOT MODIFY BLOCK <END> □

#### □ DO NOT MODIFY BLOCK <START> □
s3_bucket = BUCKET_NAME
s3_key_prefix = 'user_notebooks/'
local_directory = '/home/ec2-user/SageMaker/'

s3_client = boto3.client('s3')

def upload_notebook_to_s3(local_path, bucket, key):
    try:
        s3_client.upload_file(local_path, bucket, key)
        print(f'Successfully uploaded {local_path} to
s3://{bucket}/{key}')
    except Exception as e:
        print(f'Error uploading file: {e}')

notebook_files = [f for f in os.listdir(local_directory) if
re.match(r'.*\.ipynb$', f)]
```

```python
for notebook_file in notebook_files:
    local_path = os.path.join(local_directory, notebook_file)
    s3_key = os.path.join(s3_key_prefix, notebook_file)
    upload_notebook_to_s3(local_path, s3_bucket, s3_key)
#### □ DO NOT MODIFY BLOCK <END> □
```

End of Notebook