```
# Task 0: Install and Import Libraries
# Run this cell to install libraries:
!pip3 install "pandas==2.2.2" "scikit-learn==1.6.1" "matplotlib==3.10.0"

# ============================================================================
# Task 0: 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
# ============================================================================

### Create S3 path for the dataset
# Replace XYZXYZ with your actual random integers from your bucket name
bucket = "car-data123456"  # Change XYZXYZ to your actual bucket suffix
data_key = "car_cleaned_data/car_cleaned_data.csv"
data_location = f"s3://{bucket}/{data_key}"
###

### Load the dataset
df = pd.read_csv(data_location)
###

### Analyze the dataset
print("Dataset Shape:", df.shape)
print("\nFirst 5 rows:")
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nDataset Description:")
print(df.describe())
print("\nMissing Values:")
print(df.isnull().sum())
print("\nColumn Names:")
print(df.columns.tolist())
###

# ============================================================================
# Task 2: Feature Engineering
# ============================================================================

### Create new feature: age of the car
df['car_age'] = 2024 - df['year']
print("Car age feature created")
print(f"Car age range: {df['car_age'].min()} to {df['car_age'].max()} years")
###

### Drop the columns
df = df.drop(['car_name', 'year'], axis=1)
print("Columns 'car_name' and 'year' dropped")
print("Remaining columns:", df.columns.tolist())
###
```

```
# ================================================================================
# Task 3: Define Features and Target Variable
# ================================================================================

### Define the features and target variable
X = df.drop('selling_price', axis=1)
y = df['selling_price']
print("Features (X) shape:", X.shape)
print("Target (y) shape:", y.shape)
print("Feature columns:", X.columns.tolist())
###

# ================================================================================
# Task 4: Preprocess the Data
# ================================================================================

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

print("Numerical features:", numerical_features)
print("Categorical features:", categorical_features)
###

### Log transformation for skewed numerical features
# Apply log transformation to km_driven (assuming it's skewed)
if 'km_driven' in X.columns:
    X = X.copy()
    X['km_driven'] = np.log1p(X['km_driven'])
    print("Log transformation applied to km_driven")

# Apply log transformation to target variable (selling_price)
y = np.log1p(y)
print("Log transformation applied to selling_price")
###

# ================================================================================
# Task 5: Build a Transformer Pipeline
# ================================================================================

### Create the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)
print("Column transformer created with:")
print(f"- StandardScaler for: {numerical_features}")
print(f"- OneHotEncoder for: {categorical_features}")
###

# ================================================================================
# Task 6: Build a Model Pipeline and Split the Data
# ================================================================================

### Create a pipeline with the preprocessor and the model
model = Pipeline(
    steps=[
        ('preprocessor', preprocessor),
        ('regressor', RandomForestRegressor(random_state=8))
    ]
)
print("Model pipeline created with preprocessor and RandomForestRegressor")
```

```
###

### Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=8
)
print(f"Data split completed:")
print(f"Training set: {X_train.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")
###

# ============================================================================
# Task 7: Hyperparameter Tuning
# ============================================================================

### 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]
}
print("Parameter grid defined for hyperparameter tuning")
print("Grid will test", len(param_grid['regressor__n_estimators']) *
      len(param_grid['regressor__max_depth']) *
      len(param_grid['regressor__min_samples_split']), "combinations")
###

### Create the GridSearchCV model
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=1,
    verbose=1
)
print("GridSearchCV created with 5-fold cross-validation")
###

# ============================================================================
# Task 8: Train the Model
# ============================================================================

### Fit the grid search (this may take several minutes)
print("Starting model training with hyperparameter tuning...")
print("This may take a few minutes to complete...")
grid_search.fit(X_train, y_train)
print("Model training completed!")
###

### Best model from grid search
best_model = grid_search.best_estimator_
print("Best model extracted from grid search results")
print("Best CV score:", grid_search.best_score_)
###

# ============================================================================
# Task 9: Make Predictions
# ============================================================================

### Make predictions
y_pred = best_model.predict(X_test)
print("Predictions made on test set")
###

### Transform predictions back to original scale
```

```
y_test = np.expm1(y_test)
y_pred = np.expm1(y_pred)
print("Predictions and test values transformed back to original scale")
###

# ================================================================================
# Task 10: Evaluate the Model
# ================================================================================

### Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
###

### Print the metrics
print("="*50)
print("MODEL EVALUATION RESULTS")
print("="*50)
print(f'MAE: {mae:.2f}')
print(f'MSE: {mse:.2f}')
print(f'RMSE: {rmse:.2f}')
print(f'R2: {r2:.4f}')
print(f'Best Parameters: {grid_search.best_params_}')
print("="*50)
###

# Additional evaluation - show sample predictions
print("\nSample Predictions vs Actual Values:")
comparison = pd.DataFrame({
    'Actual_Price': y_test.iloc[:10].values,
    'Predicted_Price': y_pred[:10],
    'Difference': y_test.iloc[:10].values - y_pred[:10]
})
comparison['Accuracy_%'] = (1 - abs(comparison['Difference']) / comparison['Actual_Price']) * 100
print(comparison.round(2))

# ================================================================================
# EVALUATION AND SUBMISSION CODE - DO NOT MODIFY
# ================================================================================

# 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)]

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> 🟢
```