Microsoft Azure | databricks

Search data, notebooks, recents, and more... CTRL + P

DATABRICKS_WORKSPACE

New

Workspace
Recents
Catalog
Workflows
Compute

Data Engineering
Job Runs

Machine Learning
Playground
Experiments
Features
Models
Serving

Partner Connect

Untitled Notebook 2024-12-07 19:30:33

Python

File Edit View Run Help Last edit was 1 hour ago

Run all | cluster001 | Schedule | Share

Workspace

sanjaymass6677@gmail.com

input_data.csv
Untitled Notebook 2024-12-07 19:30:33
Untitled Notebook 2024-12-07 22:20:46

+ Code + Text

11:21 PM (2s) 1 Python

```
from azure.storage.blob import BlobServiceClient
import io
import pandas as pd

# Connection string to your Azure Blob Storage account
connection_string = "DefaultEndpointsProtocol=https;AccountName=dbstoragefordb01;
AccountKey=Vhki9bLFbWinLGmKsc+OS/jVrJ+l1dHwrgWIyr7PbmNUwa04XWljMtLxsaeRhPy4MvtzVXfHraQk+ASt6oOH+Q==;
EndpointSuffix=core.windows.net"

# Define the container name and the blob (file) name
container_name = "container01fordb"
blob_name = "cr.csv"

# Initialize the BlobServiceClient using the connection string
blob_service_client = BlobServiceClient.from_connection_string(connection_string)

# Get a BlobClient for the blob you want to read
blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)

# Download the blob's content into memory (as a stream)
blob_data = blob_client.download_blob()

# Read the content of the blob as a string (this will be CSV data)
csv_data = blob_data.readall()

# Convert the CSV data into a pandas DataFrame for easy manipulation
data_frame = pd.read_csv(io.BytesIO(csv_data))

# Print the DataFrame to see the contents
print(data_frame)
```

	product_id	product_name	...	customer_location	target_column
0	1001	Laptop	...	New York	1
1	1002	Smartphone	...	California	1
2	1003	Coffee Maker	...	Texas	0
3	1004	Headphones	...	New York	1
4	1005	Office Chair	...	California	0
5	1006	Refrigerator	...	Texas	1
6	1007	TV	...	Florida	1
7	1008	Washing Machine	...	New York	0
8	1009	Blender	...	California	1
9	1010	Gaming Console	...	Texas	1

[10 rows x 9 columns]

11:21 PM (<1s) 2

```
import pandas as pd
import numpy as np

# Assuming data_frame is your cleaned DataFrame

# 1. Cumulative Sum for 'quantity_sold'
data_frame['cumulative_quantity_sold'] = data_frame['quantity_sold'].cumsum()

# 2. Rolling Averages
# We can calculate rolling averages for numerical columns like 'price', 'quantity_sold', and
'discount'
# For simplicity, we use a 3-row rolling window for each of these columns

data_frame['rolling_avg_price'] = data_frame['price'].rolling(window=3).mean()
data_frame['rolling_avg_quantity_sold'] = data_frame['quantity_sold'].rolling(window=3).mean()
data_frame['rolling_avg_discount'] = data_frame['discount'].rolling(window=3).mean()

# 3. Date-based Features (if you have a 'date' column)
# For this example, assume the date column exists and is named 'date'
# You can extract year, month, day, weekday, etc.
# If there's no date column, you can skip this step

# Example: Assuming 'date' is a datetime column
if 'date' in data_frame.columns:
    data_frame['year'] = data_frame['date'].dt.year
    data_frame['month'] = data_frame['date'].dt.month
    data_frame['day'] = data_frame['date'].dt.day
    data_frame['weekday'] = data_frame['date'].dt.weekday

# 4. One-Hot Encoding for 'category' and 'customer_location'
# One-Hot Encoding using pandas' get_dummies
data_frame = pd.get_dummies(data_frame, columns=['category', 'customer_location'], drop_first=True)

# 5. Creating Interaction Features
```

```
# For example, the interaction between 'price' and 'quantity_sold' might be useful
data_frame['price_quantity_interaction'] = data_frame['price'] * data_frame['quantity_sold']

# 6. Feature Transformation: Log Transformation (for highly skewed features)
# Apply log transformation to 'price' if it is highly skewed
if data_frame['price'].skew() > 1:  # check if skew is significant
    data_frame['log_price'] = np.log1p(data_frame['price'])  # log1p handles 0 values safely

# Apply log transformation to 'quantity_sold' if needed
if data_frame['quantity_sold'].skew() > 1:
    data_frame['log_quantity_sold'] = np.log1p(data_frame['quantity_sold'])

# 7. Additional Custom Feature (e.g., price per unit sold)
data_frame['price_per_unit'] = data_frame['price'] / (data_frame['quantity_sold'] + 1)

# 8. Remove any unnecessary columns (optional)
# Example: If 'product_name' and 'target_column' aren't useful for modeling, drop them
data_frame = data_frame.drop(columns=['product_name', 'target_column'])

# Show the final engineered DataFrame
print(data_frame)
```

```
   product_id  price  ...  price_quantity_interaction  price_per_unit
0        1001    800  ...                       12000       50.000000
1        1002    600  ...                       18000       19.354839
2        1003    100  ...                        1000        9.090909
3        1004    150  ...                        3750        5.769231
4        1005    120  ...                        2400        5.714286
5        1006    600  ...                        7200       46.153846
6        1007    400  ...                       16000        9.756098
7        1008    700  ...                        5600       77.777778
8        1009     80  ...                        2800        2.222222
9        1010    300  ...                        6600       13.043478

[10 rows x 17 columns]
```

▶ ✓ 11:21 PM (<1s) 3

```
print(data_frame.columns)
```

```
Index(['product_id', 'price', 'quantity_sold', 'discount', 'customer_age',
       'cumulative_quantity_sold', 'rolling_avg_price',
       'rolling_avg_quantity_sold', 'rolling_avg_discount',
       'category_Furniture', 'category_Home Appliances',
       'category_Kitchenware', 'customer_location_Florida',
       'customer_location_New York', 'customer_location_Texas',
       'price_quantity_interaction', 'price_per_unit'],
      dtype='object')
```

▶ Last execution failed 4

```
# Drop rows with NaN values in the feature matrix
X = X.dropna()
y = y[X.index]  # Ensure that y matches the rows in X

# Split the data into training and testing sets again after dropping NaN rows
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

> NameError: name 'X' is not defined

Diagnose error | Debug | Assistant Quick Fix: ON

▶ Last execution failed 5

```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Impute missing values in X
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
```

```
print(f"Mean Absolute Error (MAE): {mae}")

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")

r2 = r2_score(y_test, y_pred)
print(f"R-squared (R²): {r2}")

```

> NameError: name 'X' is not defined

1

Diagnose error | Debug | Assistant Quick Fix: ON

Last execution failed 6

```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Assuming 'X' is the feature matrix and 'y' is the target variable
# Impute missing values in X
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using RMSE and R² on the test set
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae}")

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")

r2 = r2_score(y_test, y_pred)
print(f"R-squared (R²): {r2}")

# Cross-validation evaluation using RMSE and R²
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Using built-in scoring methods for cross-validation
cv_rmse_scores = cross_val_score(model, X_imputed, y, cv=kf,
scoring='neg_root_mean_squared_error')
cv_r2_scores = cross_val_score(model, X_imputed, y, cv=kf, scoring='r2')

# Convert negative RMSE to positive values
cv_rmse_scores = -cv_rmse_scores

print(f"Cross-Validation RMSE Scores: {cv_rmse_scores}")
print(f"Mean CV RMSE: {cv_rmse_scores.mean()}")

print(f"Cross-Validation R² Scores: {cv_r2_scores}")
print(f"Mean CV R²: {cv_r2_scores.mean()}")

```

> NameError: name 'X' is not defined

Diagnose error | Debug | Assistant Quick Fix: ON

11:21 PM (<1s) 7

```
print(data_frame.columns)
```

```
Index(['product_id', 'price', 'quantity_sold', 'discount', 'customer_age',
       'cumulative_quantity_sold', 'rolling_avg_price',
       'rolling_avg_quantity_sold', 'rolling_avg_discount',
       'category_Furniture', 'category_Home Appliances',
       'category_Kitchenware', 'customer_location_Florida',
       'customer_location_New York', 'customer_location_Texas',
       'price_quantity_interaction', 'price_per_unit'],
      dtype='object')
```

▶ ✓ 11:21 PM (1s) 8

```
import joblib
from azure.storage.blob import BlobServiceClient
import os
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Assuming 'X' is the feature matrix and 'y' is the target variable
# Example: Train a linear regression model (replace with your model and training logic)
# Here, I use random data as an example. Replace this with your actual dataset.
X = np.random.rand(100, 5)  # 100 samples, 5 features
y = np.random.rand(100)  # 100 target values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

# Save the trained model to a local file
model_filename = "linear_regression_model.pkl"
joblib.dump(model, model_filename)

# Azure Blob Storage details
connection_string = "DefaultEndpointsProtocol=https;AccountName=dbstoragefordb01;
AccountKey=Vhki9bLFbWinLGmKsc+OS/jVrJ+l1dHwrgWIyr7PbmNUwa04XWljMtLxsaeRhPy4MvtzVXfHraQk+ASt6oOH+Q==;
EndpointSuffix=core.windows.net"
container_name = "container01fordb"
model_blob_name = "linear_regression_model.pkl"  # Blob name in the container

# Initialize the BlobServiceClient using the connection string
blob_service_client = BlobServiceClient.from_connection_string(connection_string)

# Get the BlobClient for the model file
blob_client = blob_service_client.get_blob_client(container=container_name, blob=model_blob_name)

# Upload the model file to Azure Blob Storage
try:
    # Open the model file in binary mode and upload to Blob Storage
    with open(model_filename, "rb") as data:
        blob_client.upload_blob(data, overwrite=True)  # overwrite=True to replace any existing file

    print(f"Model successfully uploaded to Azure Blob Storage as {model_blob_name}")

    # Optionally, remove the local model file after uploading
    os.remove(model_filename)

except Exception as e:
    print(f"Error uploading model to Azure Blob Storage: {e}")
```

```
Mean Squared Error (MSE): 0.10799011571862287
Model successfully uploaded to Azure Blob Storage as linear_regression_model.pkl
```

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts