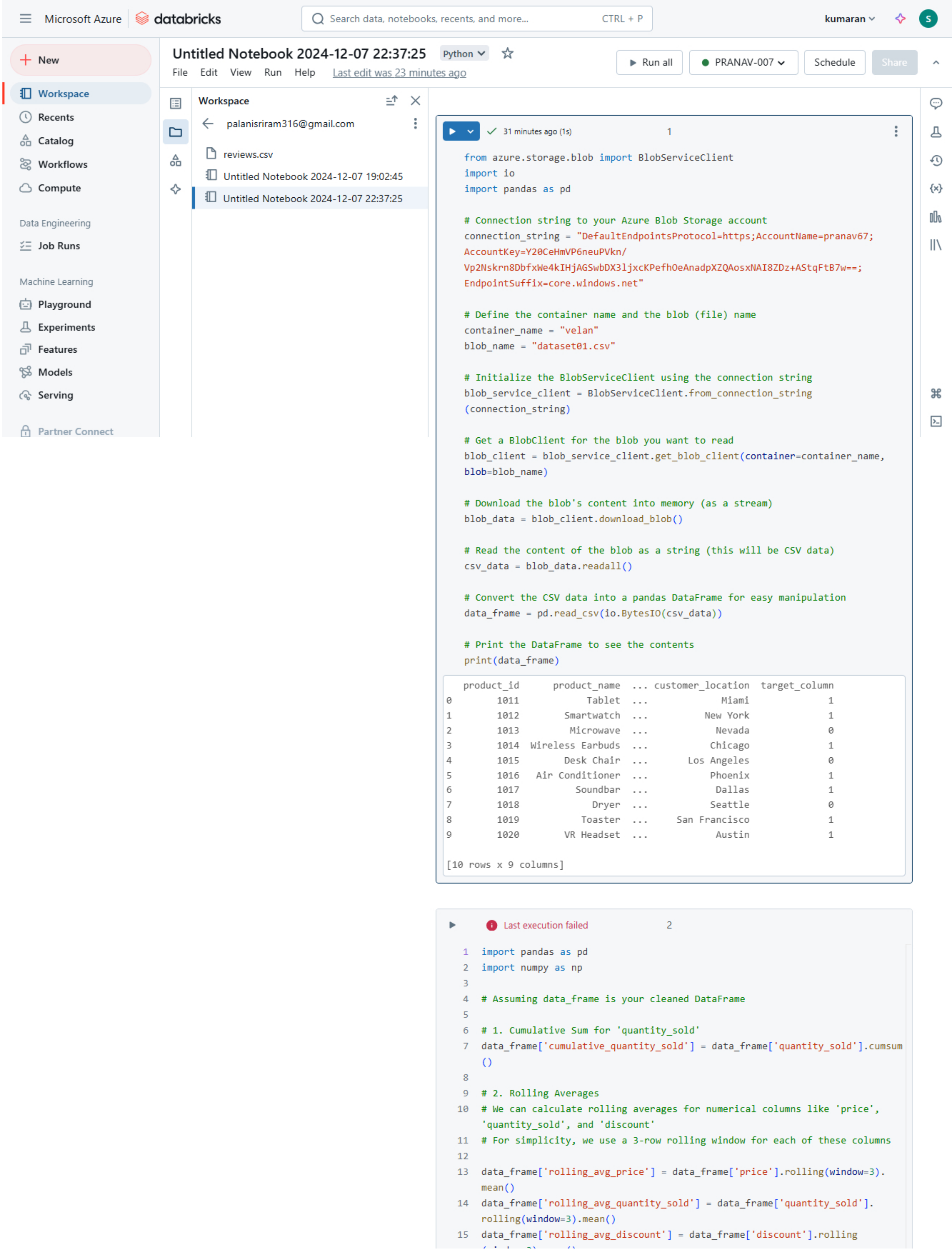


```
from azure.storage.blob import BlobServiceClient
import io
import pandas as pd

# Connection string to your Azure Blob Storage account
connection_string = "DefaultEndpointsProtocol=https;AccountName=pranav67;
AccountKey=Y20CeHmVP6neuPVkn/
Vp2Nskrn8DbfxWe4kIHjAGSwbDX3ljxcKPefhOeAnadpXZQAosxNAI8ZDz+AStqFtB7w==;
EndpointSuffix=core.windows.net"

# Define the container name and the blob (file) name
container_name = "velan"
blob_name = "dataset01.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        1011            Tablet  ...             Miami              1
1        1012        Smartwatch  ...          New York              1
2        1013         Microwave  ...            Nevada              0
3        1014  Wireless Earbuds  ...           Chicago              1
4        1015        Desk Chair  ...       Los Angeles              0
5        1016   Air Conditioner  ...           Phoenix              1
6        1017         Soundbar  ...             Dallas              1
7        1018            Dryer  ...            Seattle              0
8        1019          Toaster  ...     San Francisco              1
9        1020        VR Headset  ...            Austin              1

[10 rows x 9 columns]
```

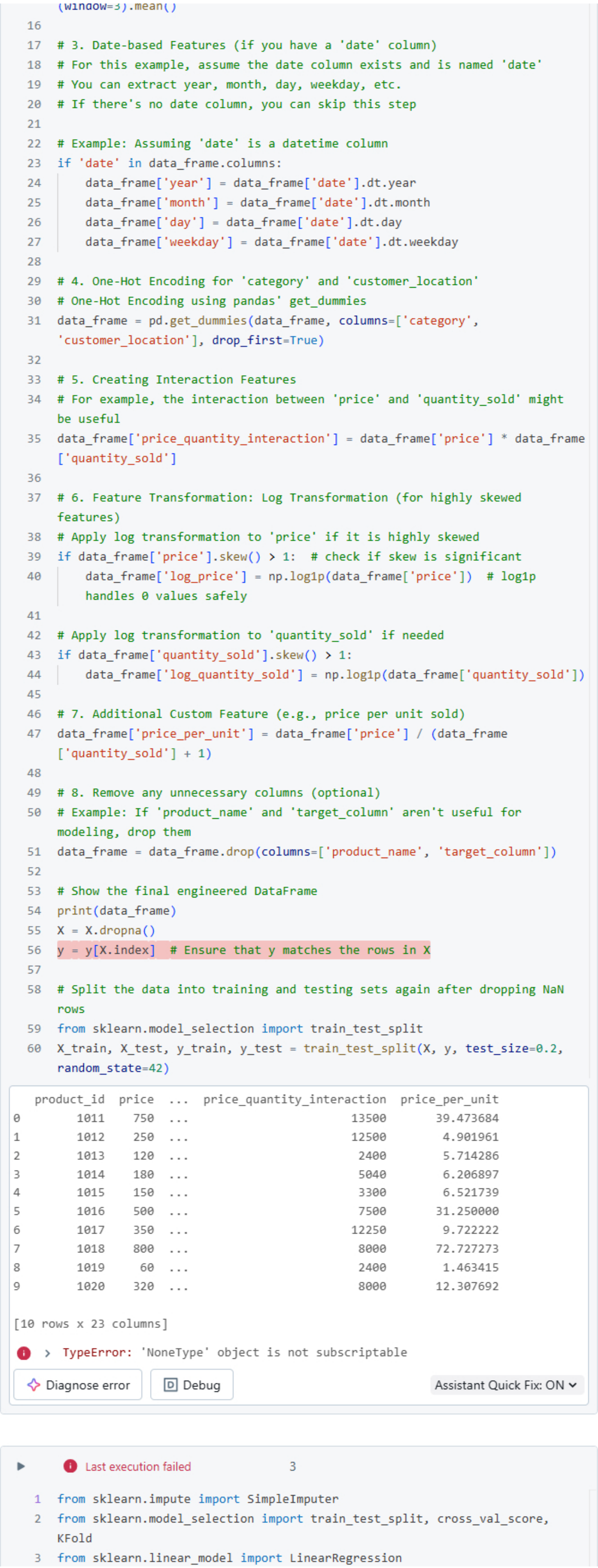
```
import pandas as pd
import numpy as np

# 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
```



```python
(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)
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)
```

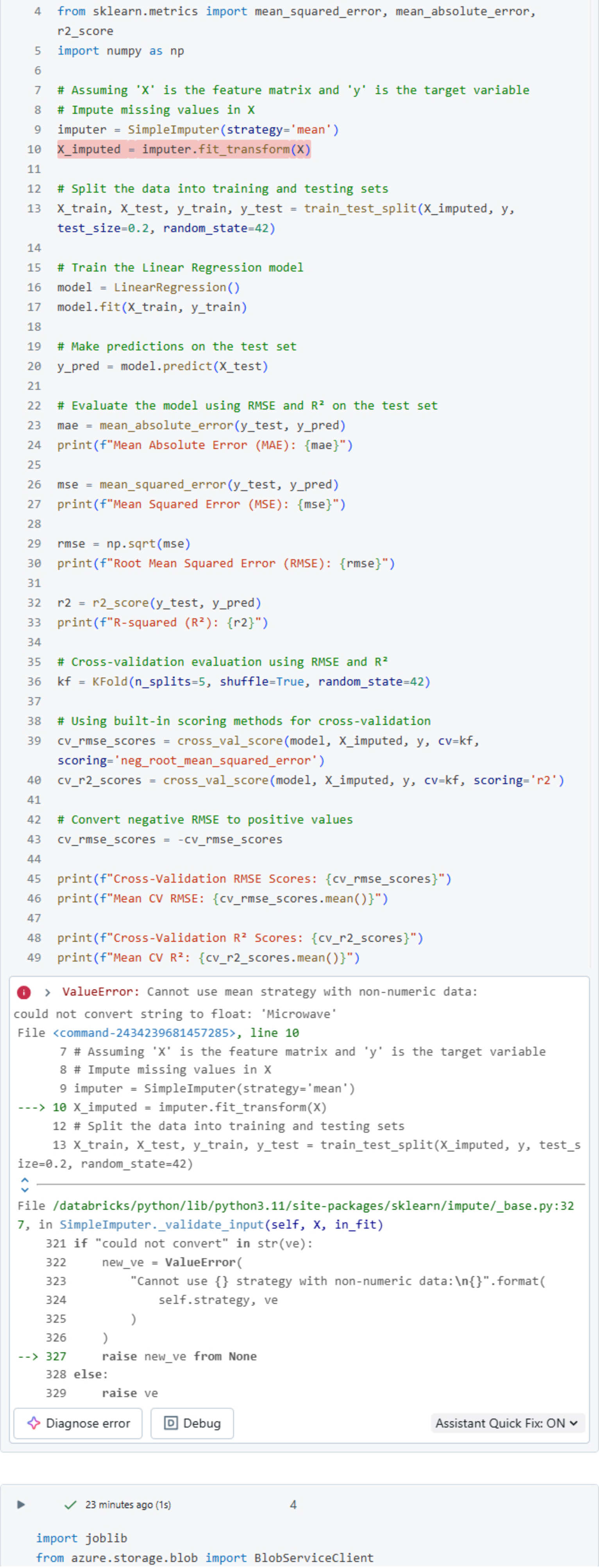
```
   product_id  price  ...  price_quantity_interaction  price_per_unit
0        1011    750  ...                       13500       39.473684
1        1012    250  ...                       12500        4.901961
2        1013    120  ...                        2400        5.714286
3        1014    180  ...                        5040        6.206897
4        1015    150  ...                        3300        6.521739
5        1016    500  ...                        7500       31.250000
6        1017    350  ...                       12250        9.722222
7        1018    800  ...                        8000       72.727273
8        1019     60  ...                        2400        1.463415
9        1020    320  ...                        8000       12.307692

[10 rows x 23 columns]
```

```
> TypeError: 'NoneType' object is not subscriptable
```

Last execution failed 3

```python
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression
```

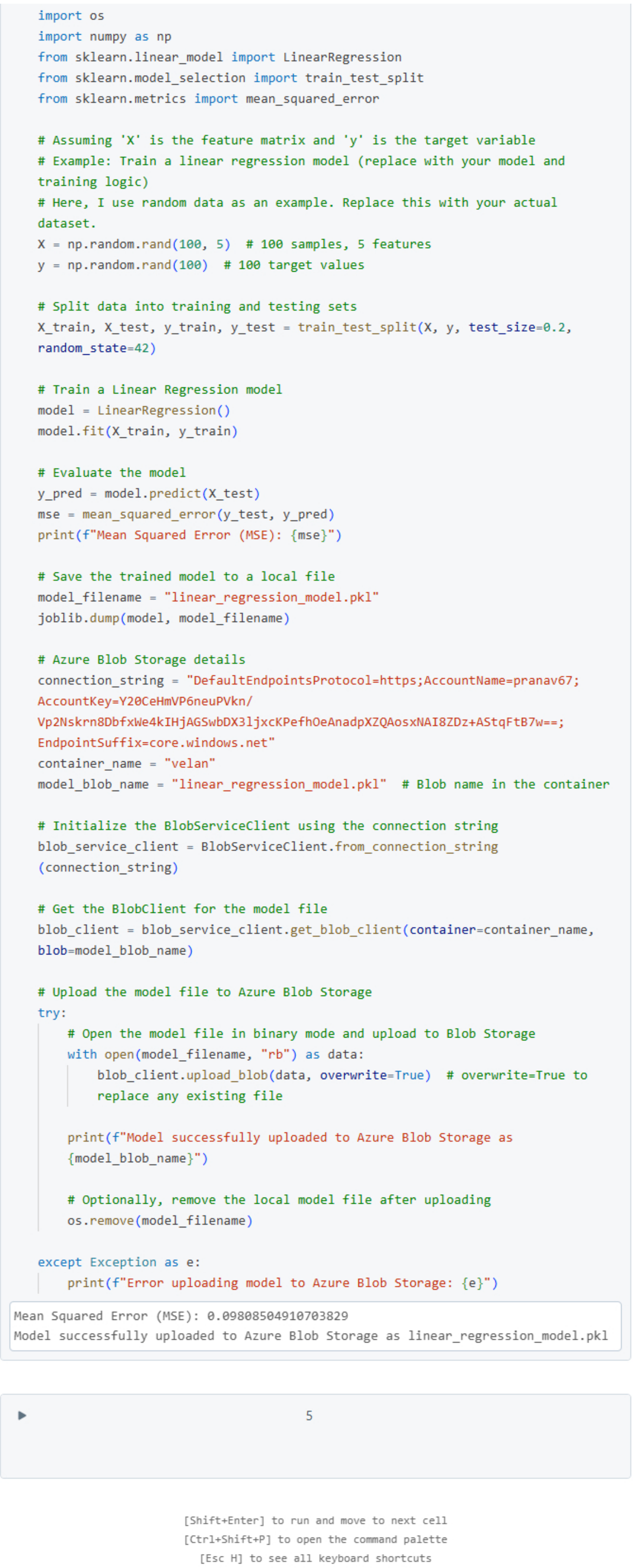


```
 4  from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
 5  import numpy as np
 6
 7  # Assuming 'X' is the feature matrix and 'y' is the target variable
 8  # Impute missing values in X
 9  imputer = SimpleImputer(strategy='mean')
10  X_imputed = imputer.fit_transform(X)
11
12  # Split the data into training and testing sets
13  X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)
14
15  # Train the Linear Regression model
16  model = LinearRegression()
17  model.fit(X_train, y_train)
18
19  # Make predictions on the test set
20  y_pred = model.predict(X_test)
21
22  # Evaluate the model using RMSE and R² on the test set
23  mae = mean_absolute_error(y_test, y_pred)
24  print(f"Mean Absolute Error (MAE): {mae}")
25
26  mse = mean_squared_error(y_test, y_pred)
27  print(f"Mean Squared Error (MSE): {mse}")
28
29  rmse = np.sqrt(mse)
30  print(f"Root Mean Squared Error (RMSE): {rmse}")
31
32  r2 = r2_score(y_test, y_pred)
33  print(f"R-squared (R²): {r2}")
34
35  # Cross-validation evaluation using RMSE and R²
36  kf = KFold(n_splits=5, shuffle=True, random_state=42)
37
38  # Using built-in scoring methods for cross-validation
39  cv_rmse_scores = cross_val_score(model, X_imputed, y, cv=kf, scoring='neg_root_mean_squared_error')
40  cv_r2_scores = cross_val_score(model, X_imputed, y, cv=kf, scoring='r2')
41
42  # Convert negative RMSE to positive values
43  cv_rmse_scores = -cv_rmse_scores
44
45  print(f"Cross-Validation RMSE Scores: {cv_rmse_scores}")
46  print(f"Mean CV RMSE: {cv_rmse_scores.mean()}")
47
48  print(f"Cross-Validation R² Scores: {cv_r2_scores}")
49  print(f"Mean CV R²: {cv_r2_scores.mean()}")
```

```
ValueError: Cannot use mean strategy with non-numeric data:
could not convert string to float: 'Microwave'
File <command-2434239681457285>, line 10
      7 # Assuming 'X' is the feature matrix and 'y' is the target variable
      8 # Impute missing values in X
      9 imputer = SimpleImputer(strategy='mean')
---> 10 X_imputed = imputer.fit_transform(X)
     12 # Split the data into training and testing sets
     13 X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_s
ize=0.2, random_state=42)

File /databricks/python/lib/python3.11/site-packages/sklearn/impute/_base.py:32
7, in SimpleImputer._validate_input(self, X, in_fit)
    321 if "could not convert" in str(ve):
    322     new_ve = ValueError(
    323         "Cannot use {} strategy with non-numeric data:\n{}".format(
    324             self.strategy, ve
    325         )
    326     )
--> 327     raise new_ve from None
    328 else:
    329     raise ve
```

```
import joblib
from azure.storage.blob import BlobServiceClient
```



```python
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=pranav67;
AccountKey=Y20CeHmVP6neuPVkn/
Vp2Nskrn8DbfxWe4kIHjAGSwbDX3ljxcKPefhOeAnadpXZQAosxNAI8ZDz+AStqFtB7w==;
EndpointSuffix=core.windows.net"
container_name = "velan"
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): 0.09808504910703829
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
```