```python
#%%
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.optimizers import Adam
#%%
# Import the datasets
csv1 = pd.read_csv('kaggle/daily-minimum-temperatures-in-me.csv')
csv2 = pd.read_csv('kaggle/monthly-beer-production-in-austr.csv')

# Convert the datasets to numpy arrays
colnames_csv1 = list(csv1.columns)
colnames_csv2 = list(csv2.columns)

print(f"Dataset 1 column names: {colnames_csv1}")
print(f"Dataset 2 column names: {colnames_csv2}")

dataset1 = csv1["Daily minimum temperatures"].astype(float).values
dataset2 = csv2["Monthly beer production"].astype(float).values

print(f"Dataset 1 shape: {dataset1.shape}")
print(f"Dataset 2 shape: {dataset2.shape}")
print(dataset1)
#%%
# Set up sliding window sizes
window_sizes = range(4, 13)  # Range from 4 to 12

# Define evaluation metric
```

```python
def evaluate_model(true, pred):
    return (np.square(true - pred)).mean(axis=0)

# Loop over the datasets
for i, dataset in enumerate([dataset1, dataset2]):
    print(f"Dataset {i+1}:")

    # Split the dataset into train and test sets
    train_size = int(len(dataset) * 0.8)  # 80% for training
    train_data, test_data = dataset[:train_size], dataset[train_size:]

    # Loop over the window sizes
    for window_size in window_sizes:
        print(f"Window Size: {window_size}")

        # Prepare the data for sliding windows
        train_X, train_y = [], []
        test_X, test_y = [], []
        for j in range(len(train_data) - window_size):
            train_X.append(train_data[j:j+window_size])
            train_y.append(train_data[j+window_size])
        for j in range(len(test_data) - window_size):
            test_X.append(test_data[j:j+window_size])
            test_y.append(test_data[j+window_size])

        # ARIMA
        arima_model = ARIMA(train_data, order=(1, 0, 0))  # Example order, modify as needed
        arima_model_fit = arima_model.fit()
        arima_predictions = arima_model_fit.forecast(steps=len(test_data))[0]
        arima_mse = evaluate_model(test_y, arima_predictions)

        # NN
        nn_model = Sequential()
        nn_model.add(Dense(10, input_dim=window_size, activation='relu'))
        nn_model.add(Dense(1))
        nn_model.compile(loss='mean_squared_error', optimizer='adam')
        nn_model.fit(np.array(train_X), np.array(train_y), epochs=50, batch_size=16, verbose=0)
        nn_predictions = nn_model.predict(np.array(test_X)).flatten()
        nn_mse = evaluate_model(test_y, nn_predictions)

        # LSTM (normal mode)
        lstm_model = Sequential()
        lstm_model.add(LSTM(10, input_shape=(1, window_size)))
        lstm_model.add(Dense(1))
        lstm_model.compile(loss='mean_squared_error', optimizer='adam')
        lstm_model.fit(np.array(train_X).reshape(-1, 1, window_size), np.array(train_y),
                       epochs=50, batch_size=16, verbose=0)
        lstm_predictions = lstm_model.predict(np.array(test_data).reshape(-1, 1, window_size)).flatten()
        lstm_mse = evaluate_model(test_data, lstm_predictions)

        # LSTM (batch mode)
        lstm_batch_model = Sequential()
        lstm_batch_model.add(LSTM(10, batch_input_shape=(16, 1, window_size), stateful=True))
        lstm_batch_model.add(Dense(1))
        lstm_batch_model.compile(loss='mean_squared_error', optimizer=Adam(lr=0.001))
        for epoch in range(50):
            lstm_batch_model.fit(np.array(train_X).reshape(-1, 1, window_size), np.array(train_y),
                                 epochs=1, batch_size=16, verbose=0, shuffle=False)
            lstm_batch_model.reset_states()
        lstm_batch_predictions = lstm_batch_model.predict(np.array(test_data).reshape(-1, 1, window_size), batch_size=16).flatten()
        lstm_batch_mse = evaluate_model(test_data, lstm_batch_predictions)

        # Print the results
        print(f"ARIMA MSE: {arima_mse}")
        print(f"NN MSE: {nn_mse}")
        print(f"LSTM (Normal) MSE: {lstm_mse}")
        print(f"LSTM (Batch) MSE: {lstm_batch_mse}")
        print("---")

#%%

```