```
1 %pip install numpy pandas river
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Collecting river
  Downloading river-0.23.0-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (9.2 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Collecting pandas
  Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (91 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 91.2/91.2 kB 3.5 MB/s eta 0:00:00
Requirement already satisfied: scipy<2.0.0,>=1.14.1 in /usr/local/lib/python3.12/dist-packages (from river) (1.16.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0
Downloading river-0.23.0-cp312-cp312-manylinux_2_28_x86_64.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 48.1 MB/s eta 0:00:00
Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 116.6 MB/s eta 0:00:00
Installing collected packages: pandas, river
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.2
    Uninstalling pandas-2.2.2:
      Successfully uninstalled pandas-2.2.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.3.3 which is incompatible.
Successfully installed pandas-2.3.3 river-0.23.0
```

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import kagglehub
6 import time
```

```
1 path = kagglehub.dataset_download("mlg-ulb/creditcardfraud")
2
3 print("Path to dataset files:", path)
```

```
Using Colab cache for faster access to the 'creditcardfraud' dataset.
Path to dataset files: /kaggle/input/creditcardfraud
```

```
1 df = pd.read_csv(path+'/creditcard.csv')
2 df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.11
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.10
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.90
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.19
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.13

5 rows × 31 columns

```
1  df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84807
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.21348
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.19435
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.32167

This notebook evaluates various online machine learning models from the `River` library for credit card fraud detection. We'll compare their performance using F1 score and MCC, considering different scaling and sampling techniques, and visualize their historical performance over time.

```
import river
from river.linear_model import LogisticRegression
from river.forest import ARFClassifier
from river.linear_model import PAClassifier
from river.ensemble import AdaBoostClassifier, ADWINBoostingClassifier
from river.tree import HoeffdingTreeClassifier
from river.imblearn import HardSamplingClassifier, RandomSampler
from river.metrics import F1, MCC
from river.preprocessing import StandardScaler
from river.stream import iter_pandas
```

```
Start coding or generate with AI.
```

Modulation

Creating a Python function `evaluate_model_pipeline` that takes a River model, its name, and the feature/target data (`X`, `y`) as input. This function will encapsulate the online learning loop, calculate F1 and MCC scores, print progress, and return the final F1 and MCC scores along with runtime.

Then, preparing a list of the following River models (including their corresponding scalers and sampling techniques, if applicable) that have been used previously in the notebook:

1. Logistic Regression (No Scaling)
2. Scaled Logistic Regression
3. Scaled Adaptive Random Forest (`ARFClassifier`)
4. Scaled Passive Aggressive Classifier (`PAClassifier`)
5. Scaled AdaBoost (with `HoeffdingTreeClassifier`)
6. `ARFClassifier` with HardSampling
7. AdaBoost with HardSampling (with Scaled `HoeffdingTreeClassifier`)
8. `ADWINBoostingClassifier` (with Scaled `LogisticRegression`)

Iterating through this list, call the `evaluate_model_pipeline` function for each model, and store the returned F1, MCC, and ROC AUC scores along with the model name. Finally, display all the collected results in a Pandas DataFrame and generate bar plots for F1 Score and MCC Score to visually compare the models.

Evaluate Model Function

```
# @title Evaluate Model Function

X_data = df.drop('Class', axis=1)
y_data = df['Class']

def evaluate_model_pipeline(model, model_name, X, y):
    """
    Evaluates a River model pipeline using online learning, calculating F1 and MCC scores.
    Stores historical scores at 10,000-row intervals.

    Args:
        model: The River model pipeline (e.g., StandardScaler() | LogisticRegression()).
        model_name (str): The name of the model for printing progress.
        X (pd.DataFrame): The feature DataFrame.
        y (pd.Series): The target Series.

    Returns:
        tuple: A tuple containing the final F1 score, final MCC score, lists of historical F1 scores, historical MCC score
    """

    # Instantiate the metrics
    f1_metric = F1()
```

```
    mcc_metric = MCC()

    # Initialize empty lists to store historical scores and row numbers
    history_f1 = []
    history_mcc = []
    row_numbers = []


    # Start timing
    start_time = time.time()

    # Iterate through the DataFrame instance by instance
    for i, (x, y_true) in enumerate(iter_pandas(X, y)):

        # Predict and update metrics
        y_pred = model.predict_one(x)

        f1_metric.update(y_true, y_pred)
        mcc_metric.update(y_true, y_pred)

        # Learn from the true label
        model.learn_one(x, y_true)

        # Store scores every 10000 rows
        if (i + 1) % 10000 == 0:
            history_f1.append(f1_metric.get())
            history_mcc.append(mcc_metric.get())
            row_numbers.append(i + 1)


    # End timing
    end_time = time.time()
    total_time = end_time - start_time

    # Print final scores
    final_f1 = f1_metric.get()
    final_mcc = mcc_metric.get()
    print(f"\nFinal Scores with {model_name}:")
    print(f"Final F1 Score = {final_f1:.4f}")
    print(f"Final MCC Score = {final_mcc:.4f}")
    print(f"Total time taken: {total_time:.2f} seconds")

```

Logistic Regression Models

Logistic Regression is a linear model for binary classification. We evaluate it with and without feature scaling, and also with HardSampling to address class imbalance.

Logistic Regression Models

```
# @title Logistic Regression Models

# Evaluate Logistic Regression (No Scaling)
model_ns_logreg = LogisticRegression()
final_f1_ns, final_mcc_ns, hist_f1_ns, hist_mcc_ns, rows_ns, time_ns = evaluate_model_pipeline(model_ns_logreg, 'Logistic


# Evaluate Scaled Logistic Regression
model_sc_logreg = StandardScaler() | LogisticRegression()
final_f1_sc, final_mcc_sc, hist_f1_sc, hist_mcc_sc, rows_sc, time_sc = evaluate_model_pipeline(model_sc_logreg, 'Scaled Lo



# Evaluate HardSampling Logistic Regression
model_hs_logreg = HardSamplingClassifier(classifier=model_sc_logreg, size=100, p=0.2, seed=42)
final_f1_hs_logreg, final_mcc_hs_logreg, hist_f1_hs_logreg, hist_mcc_hs_logreg, rows_hs_logreg, time_hs_logreg = evaluate_


```

```
# Store results for final DataFrame
results_summary = [
    {'Model': 'Logistic Regression (No Scaling)', 'Final F1 Score': final_f1_ns, 'Final MCC Score': final_mcc_ns, 'Total R
    {'Model': 'Scaled Logistic Regression', 'Final F1 Score': final_f1_sc, 'Final MCC Score': final_mcc_sc, 'Total Runtime
    {'Model': 'HardSampling Logistic Regression', 'Final F1 Score': final_f1_hs_logreg, 'Final MCC Score': final_mcc_hs_lo
]

df_summary = pd.DataFrame(results_summary)
print("\n--- Final Scores Summary ---")
```

```
Final Scores with Logistic Regression (No Scaling):
Final F1 Score = 0.0588
Final MCC Score = 0.0572
Total time taken: 14.95 seconds

Final Scores with Scaled Logistic Regression:
Final F1 Score = 0.7933
Final MCC Score = 0.7969
Total time taken: 34.13 seconds

Final Scores with HardSampling Logistic Regression:
Final F1 Score = 0.4414
Final MCC Score = 0.4525
Total time taken: 40.93 seconds

--- Final Scores Summary ---
                           Model  Final F1 Score  Final MCC Score  \
0  Logistic Regression (No Scaling)        0.058824         0.057192
1        Scaled Logistic Regression        0.793296         0.796936
2  HardSampling Logistic Regression        0.441397         0.452487

   Total Runtime
0      14.948919
1      34.126688
2      40.930604
```

Adaptive Random Forest Models

Adaptive Random Forest (ARF) is an ensemble method for online learning. It handles concept drift and works well with data streams. We evaluate it with and without scaling, and with HardSampling.

Adaptive Random Forest Models

```
# @title Adaptive Random Forest Models

# Evaluate Adaptive Random Forest (ARF) (no Scaling)
model_ARF = ARFClassifier(n_models=10, seed=42)
final_f1_ARF, final_mcc_ARF, hist_f1_ARF, hist_mcc_ARF, rows_ARF, time_ARF = evaluate_model_pipeline(model_ARF, 'Adaptive


# Evaluate Scaled Adaptive Random Forest (ARF)
model_sc_ARF = StandardScaler() | ARFClassifier(n_models=10, seed=42)
final_f1_sc_ARF, final_mcc_sc_ARF, hist_f1_sc_ARF, hist_mcc_sc_ARF, rows_sc_ARF, time_sc_ARF = evaluate_model_pipeline(mo


# Evaluate Hardsampling with Adaptive Randon Forest
base_arf = ARFClassifier(n_models=10, seed=42)
model_hs_arf = HardSamplingClassifier(classifier=base_arf, size=100, p=0.2, seed=42)

final_f1_hs_ARF, final_mcc_hs_ARF, hist_f1_hs_ARF, hist_mcc_hs_ARF, rows_hs_ARF, time_hs_ARF = evaluate_model_pipeline(mo


```

```
# Store results for the final dataframe
results_summary.extend([
    {'Model': 'Adaptive Random Forest (no Scaling)', 'Final F1 Score': final_f1_ARF, 'Final MCC Score': final_mcc_ARF, 'T
    {'Model': 'Scaled Adaptive Random Forest', 'Final F1 Score': final_f1_sc_ARF, 'Final MCC Score': final_mcc_sc_ARF, 'T
    {'Model': 'HardSampling Adaptive Random Forest', 'Final F1 Score': final_f1_hs_ARF, 'Final MCC Score': final_mcc_hs_AR
])

df_summary = pd.DataFrame(results_summary)
print("\n--- Final Scores Summary ---")
```

```
Final Scores with Adaptive Random Forest (no Scaling):
Final F1 Score = 0.5008
Final MCC Score = 0.5720
Total time taken: 512.50 seconds

Final Scores with Scaled Adaptive Random Forest:
Final F1 Score = 0.4102
Final MCC Score = 0.4965
Total time taken: 460.02 seconds

Final Scores with HardSampling Adaptive Random Forest:
Final F1 Score = 0.7371
Final MCC Score = 0.7534
Total time taken: 565.80 seconds

--- Final Scores Summary ---
                              Model  Final F1 Score  Final MCC Score  \
3  Adaptive Random Forest (no Scaling)        0.500754         0.571952
4        Scaled Adaptive Random Forest        0.410175         0.496511
5  HardSampling Adaptive Random Forest        0.737101         0.753400

   Total Runtime
3     512.502507
4     460.019537
5     565.795731
```

Passive Aggressive Classifier Models

Passive Aggressive Classifier (PAClassifier) is a simple yet effective online learning algorithm that updates the model only when a misclassification occurs. We evaluate it with and without scaling, and with HardSampling.

PassiveAgressive Classifier Models

```
# @title PassiveAgressive Classifier Models

#Evaluate PassiveAgressive Classifier (no Scaling)
model_pa = PAClassifier(C=0.01, mode=1)
final_f1_pa, final_mcc_pa, hist_f1_pa, hist_mcc_pa, rows_pa, time_pa = evaluate_model_pipeline(model_pa, 'Passive Agressiv



# Evaluate Scaled PassiveAgressive Classifier
model_sc_pa = StandardScaler() | PAClassifier(C=0.01, mode=1)
final_f1_sc_pa, final_mcc_sc_pa, hist_f1_sc_pa, hist_mcc_sc_pa, rows_sc_pa, time_sc_pa = evaluate_model_pipeline(model_sc_



# Evaluate HardSampling with PassiveAgressive Classifiers
base_pa = StandardScaler() | PAClassifier(C=0.01, mode=1)
model_hs_pa = HardSamplingClassifier(classifier=base_pa, size=100, p=0.2, seed=42)

final_f1_hs_pa, final_mcc_hs_pa, hist_f1_hs_pa, hist_mcc_hs_pa, rows_hs_pa, time_hs_pa = evaluate_model_pipeline(model_hs_

```

```
21
22
23 # Store results for the final dataframe
24 results_summary.extend([
25     {'Model': 'Passive Agressive Classifier (no Scaling)', 'Final F1 Score': final_f1_pa, 'Final MCC Score': final_mcc_pa
26     {'Model': 'Scaled Passive Agressive Classifier', 'Final F1 Score': final_f1_sc_pa, 'Final MCC Score': final_mcc_sc_pa
27     {'Model': 'HardSampling Passive Agressive Classifier', 'Final F1 Score': final_f1_hs_pa, 'Final MCC Score': final_mcc
28 ])
29
30 df_summary = pd.DataFrame(results_summary)
31 print("\n--- Final Scores Summary ---")
32 print(df_summary.iloc[6:9])
```

```
Final Scores with Passive Agressive Classifier (no Scaling):
Final F1 Score = 0.0528
Final MCC Score = 0.0512
Total time taken: 11.48 seconds

Final Scores with Scaled Passive Agressive Classifier:
Final F1 Score = 0.8000
Final MCC Score = 0.8020
Total time taken: 28.52 seconds

Final Scores with HardSampling Passive Agressive Classifier:
Final F1 Score = 0.5483
Final MCC Score = 0.5972
Total time taken: 39.36 seconds

--- Final Scores Summary ---
                                    Model  Final F1 Score  Final MCC Score  \
6  Passive Agressive Classifier (no Scaling)        0.052830         0.051206
7        Scaled Passive Agressive Classifier        0.800000         0.801971
8  HardSampling Passive Agressive Classifier        0.548295         0.597199

   Total Runtime
6      11.480768
7      28.523685
8      39.362953
```

AdaBoost Classifier Models

AdaBoost (Adaptive Boosting) is an ensemble meta-algorithm that can be used with a variety of other learning algorithms to improve performance. We use it with Logistic Regression, Hoeffding Trees, and Passive Aggressive Classifiers as base estimators.

AdaBoost Classifier Models

```
1 # @title AdaBoost Classifier Models
2
3 # Evaluate AdaBoost Logistic Regression
4 model_ada = AdaBoostClassifier(model=StandardScaler() | LogisticRegression(), n_models = 10, seed=42)
5 final_f1_ada, final_mcc_ada, hist_f1_ada, hist_mcc_ada, rows_ada, time_ada = evaluate_model_pipeline(model_ada, 'AdaBoost
6
7
8 # Evaluate AdaBoost PassiveAgressive Classifier
9 model_ada_pa = AdaBoostClassifier(model=StandardScaler() | PAClassifier(C=0.1, mode=1), n_models = 10, seed=42)
10 final_f1_ada_pa, final_mcc_ada_pa, hist_f1_ada_pa, hist_mcc_ada_pa, rows_ada_pa, time_ada_pa = evaluate_model_pipeline(mo
11
12
13 # Store scores in the result summary
14 results_summary.extend([
15     {'Model': 'AdaBoost Logistic Regression', 'Final F1 Score': final_f1_ada, 'Final MCC Score': final_mcc_ada, 'Total Ru
16     {'Model': 'AdaBoost PassiveAgressive Classifier', 'Final F1 Score': final_f1_ada_pa, 'Final MCC Score': final_mcc_ada
17 ])
```

```
18
19 df_summary = pd.DataFrame(results_summary)
20 print("\n--- Final Scores Summary ---")
```

```
Final Scores with AdaBoost Logistic Regression:
Final F1 Score = 0.7404
Final MCC Score = 0.7400
Total time taken: 246.54 seconds

Final Scores with AdaBoost PassiveAgressive Classifier:
Final F1 Score = 0.7518
Final MCC Score = 0.7514
Total time taken: 216.25 seconds

--- Final Scores Summary ---
                                Model  Final F1 Score  Final MCC Score  \
9           AdaBoost Logistic Regression        0.740443         0.740027
10  AdaBoost PassiveAgressive Classifier        0.751788         0.751370

    Total Runtime
9      246.536836
10     216.249747
```

```
 1 # Evaluate AdaBoost Hoeffding Tree
 2 model_ada_ht = AdaBoostClassifier(model=StandardScaler() | HoeffdingTreeClassifier(), n_models = 10, seed=42)
 3 final_f1_ada_ht, final_mcc_ada_ht, hist_f1_ada_ht, hist_mcc_ada_ht, rows_ada_ht, time_ada_ht = evaluate_model_pipeline(mode
 4
 5 results_summary.extend([
 6     {'Model': 'AdaBoost Hoeffding Tree', 'Final F1 Score': final_f1_ada_ht, 'Final MCC Score': final_mcc_ada_ht, 'Total Rur
 7 ])
 8
 9 df_summary = pd.DataFrame(results_summary)
10 print("\n--- Final Scores Summary ---")
11 print(df_summary.iloc[11:12])
```

```
Final Scores with AdaBoost Hoeffding Tree:
Final F1 Score = 0.7903
Final MCC Score = 0.7929
Total time taken: 972.22 seconds

--- Final Scores Summary ---
                   Model  Final F1 Score  Final MCC Score  Total Runtime
11  AdaBoost Hoeffding Tree        0.790287         0.792909     972.216348
```

```
1 Start coding or generate with AI.
```

ADWINBoosting Classifier Models

ADWINBoostingClassifier is an ensemble method that combines AdaBoost with the Adaptive Windowing (ADWIN) algorithm, allowing it to adapt to concept drift. We evaluate it with Logistic Regression and Passive Aggressive Classifiers as base estimators.

```
 1 # Evaluate ADWINBoosting with Logistic Regression
 2 model_adwin_lr = ADWINBoostingClassifier(model=StandardScaler() | LogisticRegression(), n_models=10,seed=42)
 3 final_f1_adwin_lr, final_mcc_adwin_lr, hist_f1_adwin_lr, hist_mcc_adwin_lr, rows_adwin_lr, time_adwin_lr = evaluate_model_
 4
 5 # Evaluate ADWINBoosting with PassiveAgressive Classifier
 6 model_adwin_pa = ADWINBoostingClassifier(model=StandardScaler() | PAClassifier(C=0.1, mode=1), n_models=10,seed=42)
 7 final_f1_adwin_pa, final_mcc_adwin_pa, hist_f1_adwin_pa, hist_mcc_adwin_pa, rows_adwin_pa, time_adwin_pa = evaluate_model_
 8
 9
```

```
# Store the results
results_summary.extend([
    {'Model': 'ADWINBoosting Logistic Regression', 'Final F1 Score': final_f1_adwin_lr, 'Final MCC Score': final_mcc_adwi
    {'Model': 'ADWINBoosting PassiveAgressive Classifier', 'Final F1 Score': final_f1_adwin_pa, 'Final MCC Score': final_
])

df_summary = pd.DataFrame(results_summary)
print("\n--- Final Scores Summary ---")
```

```
Final Scores with ADWINBoosting Logistic Regression:
Final F1 Score = 0.7955
Final MCC Score = 0.7997
Total time taken: 427.97 seconds

Final Scores with ADWINBoosting PassiveAgressive Classifier:
Final F1 Score = 0.7893
Final MCC Score = 0.7900
Total time taken: 430.26 seconds

--- Final Scores Summary ---
                                    Model  Final F1 Score  \
12          ADWINBoosting Logistic Regression        0.795506
13  ADWINBoosting PassiveAgressive Classifier        0.789305

    Final MCC Score  Total Runtime
12         0.799676     427.972467
13         0.790049     430.263999
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Results

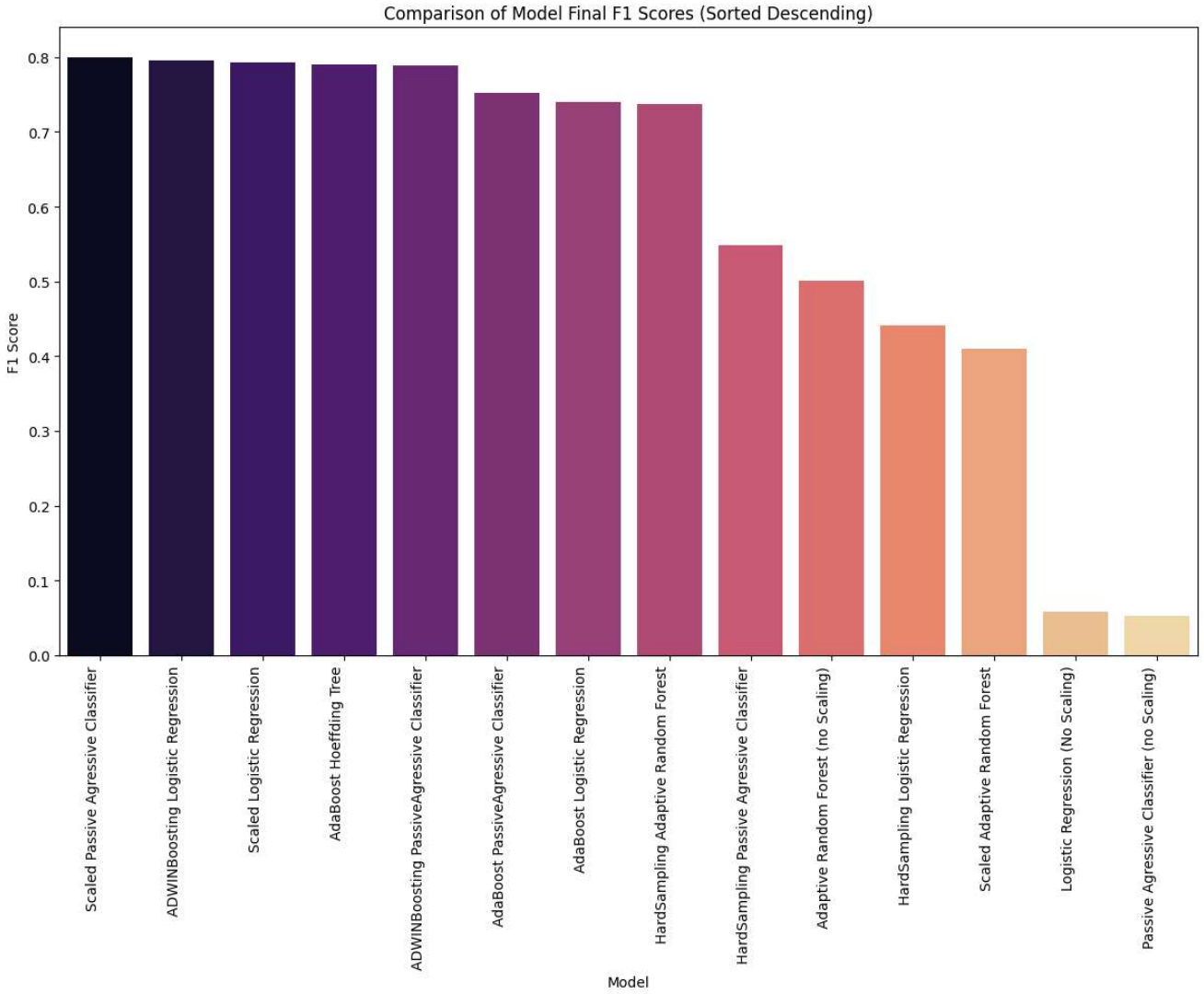
```
df_summary.round(4)
```

	Model	Final F1 Score	Final MCC Score	Total Runtime
0	Logistic Regression (No Scaling)	0.0588	0.0572	14.9489
1	Scaled Logistic Regression	0.7933	0.7969	34.1267
2	HardSampling Logistic Regression	0.4414	0.4525	40.9306
3	Adaptive Random Forest (no Scaling)	0.5008	0.5720	512.5025
4	Scaled Adaptive Random Forest	0.4102	0.4965	460.0195
5	HardSampling Adaptive Random Forest	0.7371	0.7534	565.7957
6	Passive Agressive Classifier (no Scaling)	0.0528	0.0512	11.4808
7	Scaled Passive Agressive Classifier	0.8000	0.8020	28.5237
8	HardSampling Passive Agressive Classifier	0.5483	0.5972	39.3630
9	AdaBoost Logistic Regression	0.7404	0.7400	246.5368
10	AdaBoost PassiveAgressive Classifier	0.7518	0.7514	216.2497
11	AdaBoost Hoeffding Tree	0.7903	0.7929	972.2163
12	ADWINBoosting Logistic Regression	0.7955	0.7997	427.9725
13	ADWINBoosting PassiveAgressive Classifier	0.7893	0.7900	430.2640

```
# Set the index to model names for easier plotting
```

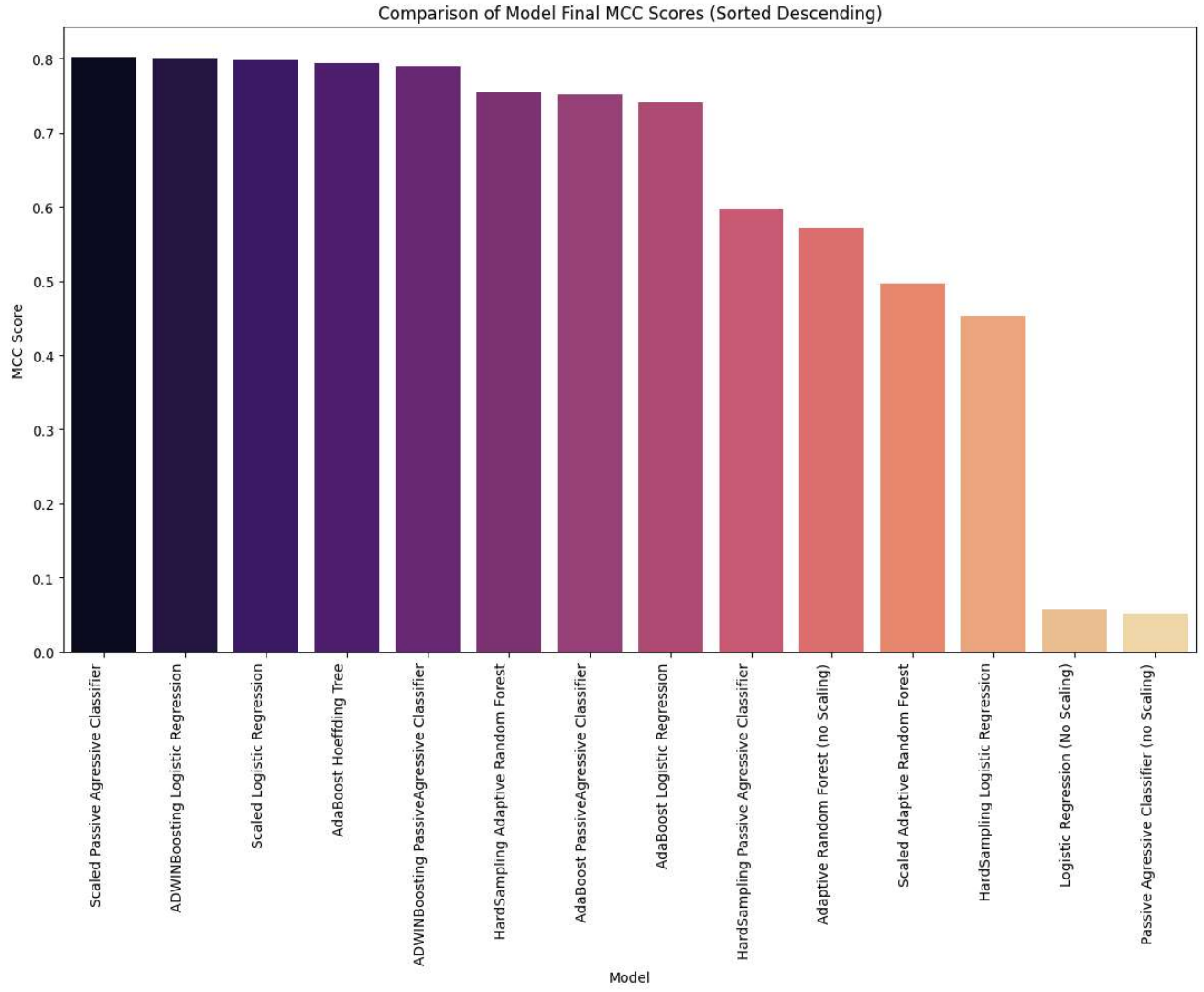
```
df_results = df_summary.set_index('Model')

# Plotting the scores
df_f1_sorted = df_results.sort_values(by='Final F1 Score', ascending=False)

# F1 Score plot
plt.figure(figsize=(12, 10))
sns.barplot(x=df_f1_sorted.index, y='Final F1 Score', data=df_f1_sorted,
palette='magma')
plt.title('Comparison of Model Final F1 Scores (Sorted Descending)')
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
plt.show()


```

```
/tmp/ipython-input-3445809527.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=df_f1_sorted.index, y='Final F1 Score', data=df_f1_sorted, palette='magma')
```



```
df_mcc_sorted = df_results.sort_values(by='Final MCC Score', ascending=False)

# MCC Score plot
plt.figure(figsize=(12, 10))
sns.barplot(x=df_mcc_sorted.index, y='Final MCC Score', data=df_mcc_sorted, palette='magma')
plt.title('Comparison of Model Final MCC Scores (Sorted Descending)')
plt.xlabel('Model')
plt.ylabel('MCC Score')
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
plt.show()
```

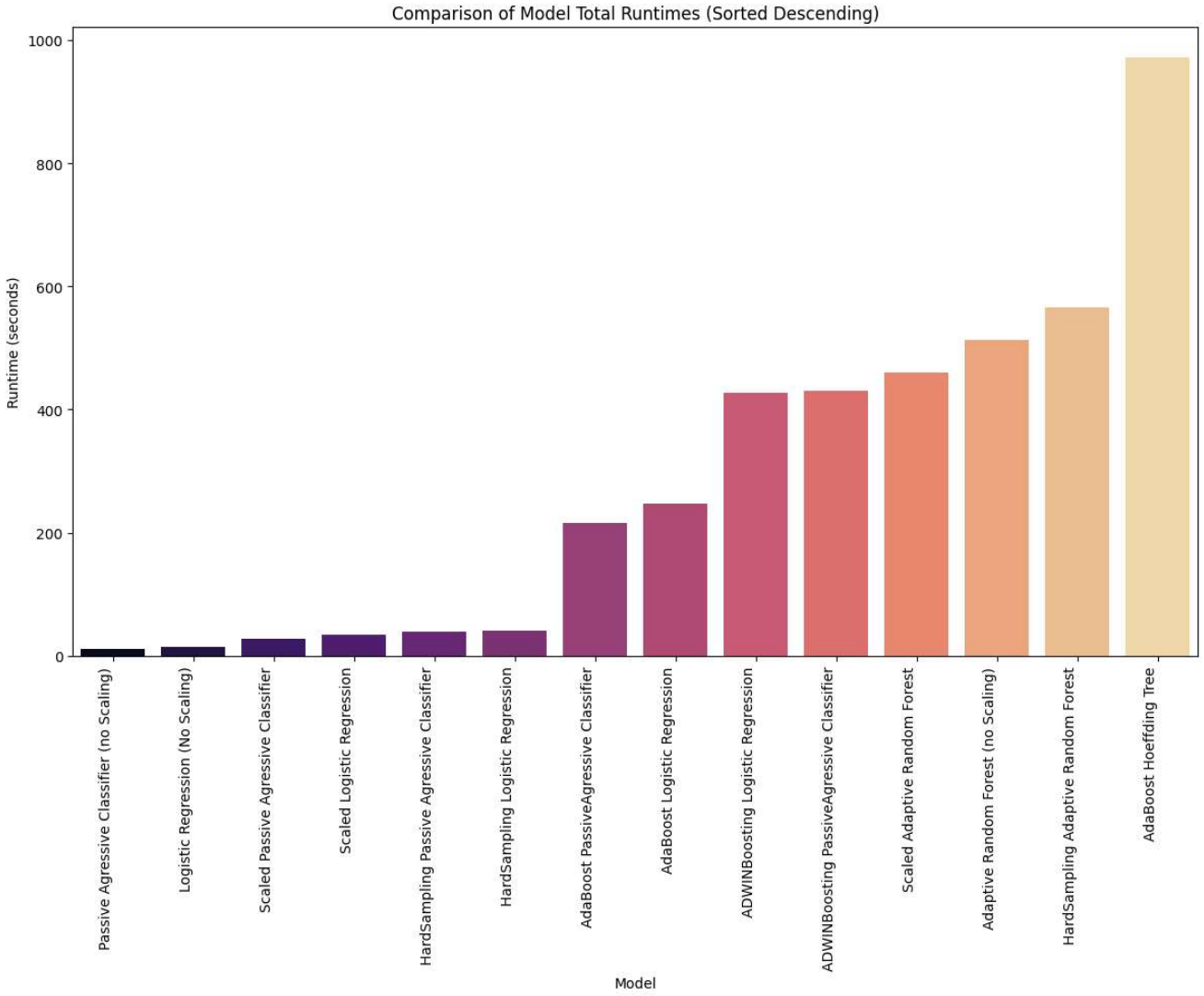
```
/tmp/ipython-input-3174712153.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=df_mcc_sorted.index, y='Final MCC Score', data=df_mcc_sorted, palette='magma')
```




```
df_runtime_sorted = df_results.sort_values(by='Total Runtime', ascending=True)

# Total Runtime plot
plt.figure(figsize=(12, 10))
sns.barplot(x=df_runtime_sorted.index, y='Total Runtime', data=df_runtime_sorted, palette='magma')
```

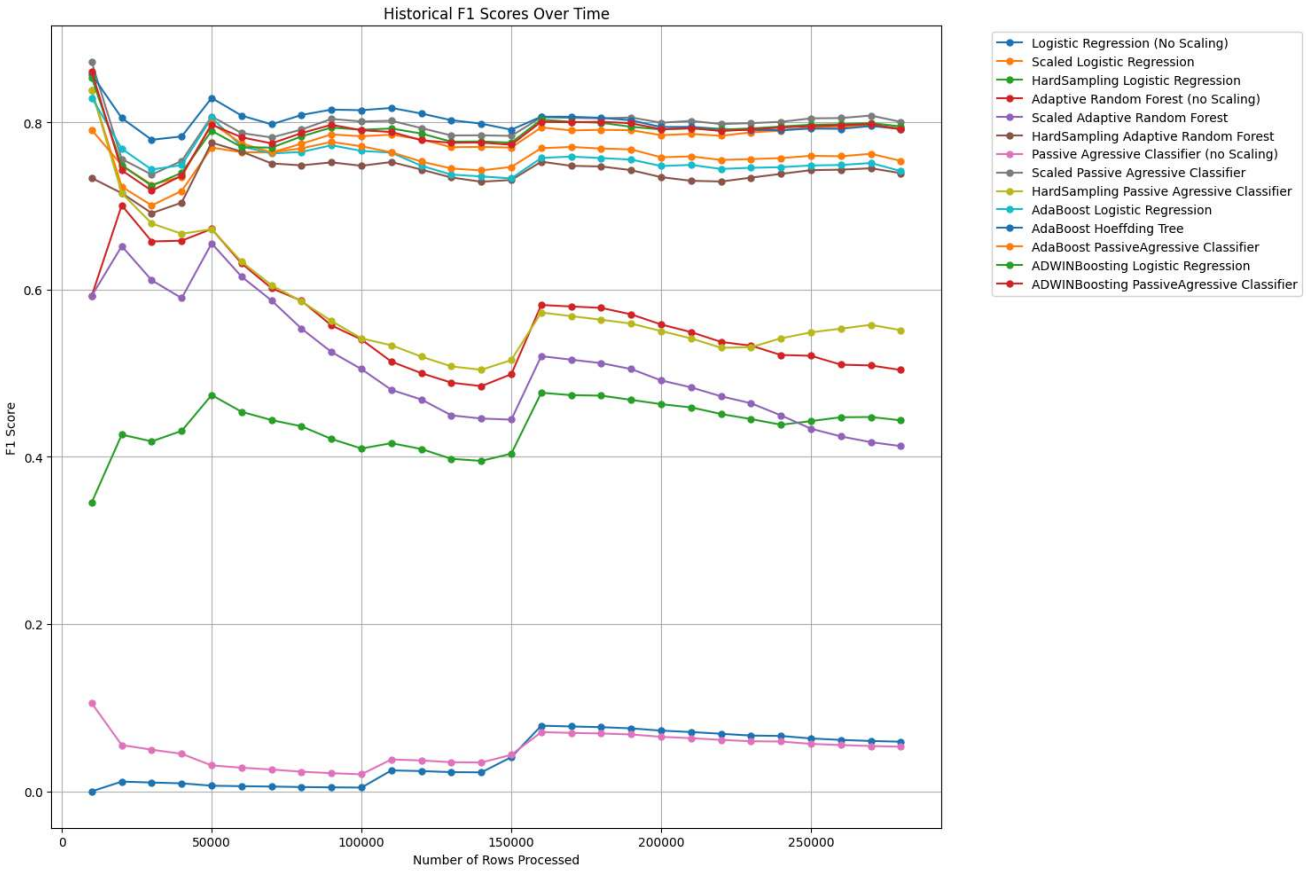
```
plt.title('Comparison of Model Total Runtimes (Sorted Descending)')
plt.xlabel('Model')
plt.ylabel('Runtime (seconds)')
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
```

```
/tmp/ipython-input-3676103878.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=df_runtime_sorted.index, y='Total Runtime', data=df_runtime_sorted, palette='magma')
```

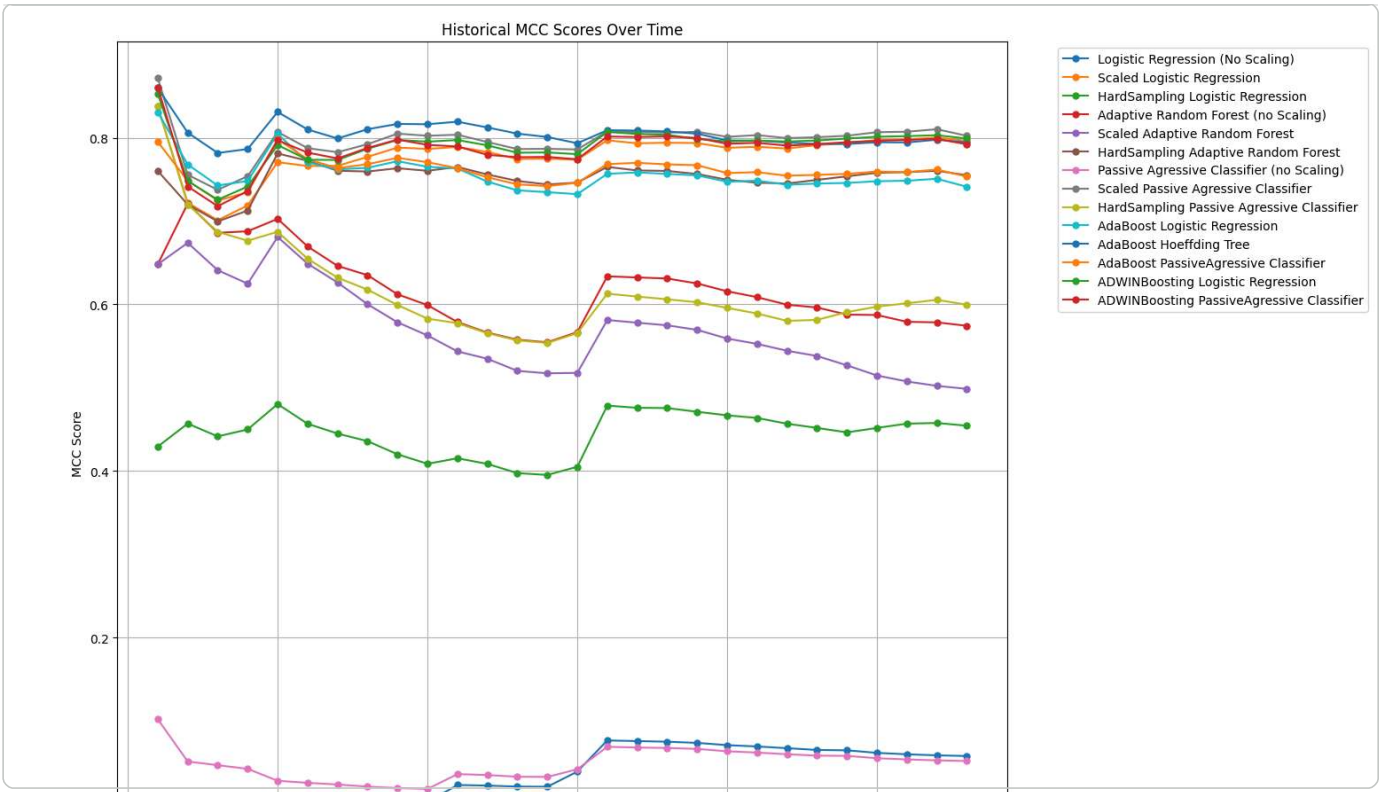



```
import matplotlib.pyplot as plt
import seaborn as sns

# Collect all historical data
historical_data = []

historical_data.extend([
    {'Model': 'Logistic Regression (No Scaling)', 'f1_history': hist_f1_ns, 'mcc_history': hist_mcc_ns, 'rows': rows_ns},
    {'Model': 'Scaled Logistic Regression', 'f1_history': hist_f1_sc, 'mcc_history': hist_mcc_sc, 'rows': rows_sc},
    {'Model': 'HardSampling Logistic Regression', 'f1_history': hist_f1_hs_logreg, 'mcc_history': hist_mcc_hs_logreg, 'rou
    {'Model': 'Adaptive Random Forest (no Scaling)', 'f1_history': hist_f1_ARF, 'mcc_history': hist_mcc_ARF, 'rows': rows_
```

```
    {'Model': 'Scaled Adaptive Random Forest', 'f1_history': hist_f1_sc_ARF, 'mcc_history': hist_mcc_sc_ARF, 'rows': rows_
    {'Model': 'HardSampling Adaptive Random Forest', 'f1_history': hist_f1_hs_ARF, 'mcc_history': hist_mcc_hs_ARF, 'rows'
    {'Model': 'Passive Agressive Classifier (no Scaling)', 'f1_history': hist_f1_pa, 'mcc_history': hist_mcc_pa, 'rows': r
    {'Model': 'Scaled Passive Agressive Classifier', 'f1_history': hist_f1_sc_pa, 'mcc_history': hist_mcc_sc_pa, 'rows': r
    {'Model': 'HardSampling Passive Agressive Classifier', 'f1_history': hist_f1_hs_pa, 'mcc_history': hist_mcc_hs_pa, 'rc
    {'Model': 'AdaBoost Logistic Regression', 'f1_history': hist_f1_ada, 'mcc_history': hist_mcc_ada, 'rows': rows_ada},
    {'Model': 'AdaBoost Hoeffding Tree', 'f1_history': hist_f1_ada_ht, 'mcc_history': hist_mcc_ada_ht, 'rows': rows_ada_ht
    {'Model': 'AdaBoost PassiveAgressive Classifier', 'f1_history': hist_f1_ada_pa, 'mcc_history': hist_mcc_ada_pa, 'rows
    {'Model': 'ADWINBoosting Logistic Regression', 'f1_history': hist_f1_adwin_lr, 'mcc_history': hist_mcc_adwin_lr, 'rows
    {'Model': 'ADWINBoosting PassiveAgressive Classifier', 'f1_history': hist_f1_adwin_pa, 'mcc_history': hist_mcc_adwin_p
])

# Plotting Historical F1 Scores
plt.figure(figsize=(15, 10))
for data in historical_data:
    plt.plot(data['rows'], data['f1_history'], label=data['Model'], marker='o', markersize=5)
plt.title('Historical F1 Scores Over Time')
plt.xlabel('Number of Rows Processed')
plt.ylabel('F1 Score')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()

```




```
# Plotting Historical MCC Scores
plt.figure(figsize=(15, 10))
for data in historical_data:
    plt.plot(data['rows'], data['mcc_history'], label=data['Model'],
    marker='o', markersize=5)
plt.title('Historical MCC Scores Over Time')
plt.xlabel('Number of Rows Processed')
plt.ylabel('MCC Score')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Conclusion

This notebook systematically evaluated various online machine learning models from the `River` library for credit card fraud detection. The models were compared based on their final F1 score, MCC score, and total runtime, with historical performance also visualized.

Key Findings:

- **Impact of Scaling:** Feature scaling (e.g., using `StandardScaler`) had a significant positive impact on the performance of linear models like `LogisticRegression` and `PAClassifier`. Models without scaling often performed poorly, highlighting the importance of preprocessing for these algorithms in an online learning context.
- **Impact of HardSampling:** While `HardSamplingClassifier` was intended to address class imbalance, its effect on overall performance (F1 and MCC) varied. For Logistic Regression, HardSampling drastically reduced performance compared to its scaled counterpart. For `ARFClassifier` and `PAClassifier`, HardSampling generally improved performance, indicating its utility can be model-specific and might require careful tuning.
- **Ensemble Methods:** Ensemble methods like `AdaBoostClassifier` and `ADWINBoostingClassifier` generally showed strong performance. `ADWINBoostingClassifier` with `LogisticRegression` as a base estimator achieved one of the highest F1 and MCC scores, demonstrating its robustness and adaptability to data streams.
- **Performance vs. Runtime:** There was often a trade-off between model complexity, performance, and runtime. Simple scaled models like `Scaled Passive Agressive Classifier` and `Scaled Logistic Regression` offered competitive performance with relatively lower runtimes compared to more complex ensemble models like `AdaBoost Hoeffding Tree`.
- **Top Performing Models:**
    - `Scaled Logistic Regression` and `Scaled Passive Agressive Classifier` consistently delivered high F1 and MCC