FireDucks:
Diving into API Compatibility with Pandas

In past discussions, I've shed light on FireDucks, briefly touching on its capabilities and innovative approach to DataFrame manipulation (see Blog Article #1 link, will be updated in final design). I've also put it to test against Polars, where FireDucks demonstrated its prowess by outshining Polars in performance (Blog Article #2 link, will be updated in final design). FireDucks boasts a feature that has piqued the interest of many: its touted high compatibility with the Pandas API.

Now, we dive into the world of FireDucks, a rising star in the field of Dataframe manipulation. With a commitment to enhanced performance and unique features, FireDucks stands out among its peers. One of its most alluring aspects for users is its seamless integration with Pandas, ensuring a smooth transition and effortless utilization of existing code.
Join us as we explore FireDucks' API similarities and differences compared to Pandas, unravelling its potential to revolutionize data manipulation workflows.

Similarities for a Familiar Experience:

FireDucks aims to provide a familiar environment for Pandas users by incorporating:

- Core Data Structures: It utilizes Series and DataFrames, mimicking the fundamental building blocks of Pandas data manipulation.
- Common Operations: Many frequently used functions in Pandas have counterparts in FireDucks which can be used without any change in syntax.

These include:

groupby · filter · sort · head · tail · merge · join · at · iat · ioc

and many more

There are two ways to run Pandas code under FireDucks:

- Execute the code using an import hook without making any change in Pandas code:
  `python -mfireducks.imhook sample.py`
- Changing the import statement, in this case no import hook is required:
  Use 'import fireducks.pandas as pd' instead of 'import pandas as pd'

Here is a comparison of outputs from Pandas and Fireducks.

For Fireducks all the Pandas code has been executed as it is with the import hook for FireDucks

```
#Sample data
data = {'category': ['A', 'A', 'B', 'B', 'C'],
    'value': [10, 15, 20, 25, 30]}
```

1 #groupby example

```
1   #Pandas code
3   import pandas as pd
4   #create dataframe from data
5   df = pd.DataFrame(data)
6
7   #category wise mean and standard deviation calculation
9   grouped_by_category_stats = df.groupby('category').agg({'value': ['mean', 'std']})
10
```

	value	
	mean	std
category		
A	12.5	3.535534
B	22.5	3.535534
C	30.0	NaN

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

	value	
	mean	std
category		
A	12.5	3.535534
B	22.5	3.535534
C	30.0	NaN

2 #filter example

```
1   #Pandas code
3   import pandas as pd
4
5   #create dataframe from data
6   df = pd.DataFrame(data)
7
9   #filtering where category is 'B' and 'value' is greater than 20
10  print ((df['category'] == 'B') & (df['value'] > 20))
```

```
0    False
1    False
2    False
3     True
4    False
dtype: bool
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
0    False
1    False
2    False
3     True
4    False
dtype: bool
```

3 #sorting example

```
1   #Pandas code
3   import pandas as pd
4
5   #create dataframe from data
6   df = pd.DataFrame(data)
7
9   #sorting by 'value' column
10  print (df.sort_values(by='value'))
```

	category	value
0	A	10
1	A	15
2	B	20
3	B	25
4	C	30

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

	category	value
0	A	10
1	A	15
2	B	20
3	B	25
4	C	30

4 #head example

```
1   #Pandas code
3   import pandas as pd
4
5   #create dataframe from data
6   df = pd.DataFrame(data)
7
9   #show first three rows
10  print (df.head(3))
```

	category	value
0	A	10
1	A	15
2	B	20

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

	category	value
0	A	10
1	A	15
2	B	20

5 #tail example

```
1    #Pandas code
3    import pandas as pd
4
5    #create dataframe from data
6    df = pd.DataFrame(data)
7
9    #show last three rows
10   print (df.tail(3))
```

```
  category  value
2        B     20
3        B     25
4        C     30
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
  category  value
2        B     20
3        B     25
4        C     30
```

6 #merge example

```
1    #Pandas code
3    import pandas as pd
4
5    #Dataframe with 'ID' and 'Name'
6    df1 = pd.DataFrame({
7      'ID': [1, 2, 3],
9      'Name': ['Alice', 'Bob', 'Charlie']
10   })
11
12   #Dataframe with 'ID' and 'Age'
13   df2 = pd.DataFrame({
14     'ID': [2, 3, 4],
15     'Age': [25, 30, 22]
16   })
17
```

```
DataFrame 1:
   ID      Name
0   1     Alice
1   2       Bob
2   3   Charlie

DataFrame 2:
   ID  Age
0   2   25
1   3   30
2   4   22
```

```
# Merging the DataFrames on the 'ID' column
merged_df = pd.merge(df1, df2, on='ID')

# Displaying the merged DataFrame
print("Merged DataFrame:")
print(merged_df)
```

```
Merged DataFrame:
   ID      Name  Age
0   2       Bob   25
1   3   Charlie   30
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
Merged DataFrame:
   ID      Name  Age
0   2       Bob   25
1   3   Charlie   30
```

7 #join example

```
1    #Pandas code
3    import pandas as pd
4
5    # Creating two sample DataFrames with common indices
6    df1 = pd.DataFrame({
7      'Name': ['Alice', 'Bob', 'Charlie'],
9    }, index=[1, 2, 3])
10
11   df2 = pd.DataFrame({
12     'Age': [25, 30, 22],
13   }, index=[2, 3, 4])
```

```
DataFrame 1:
      Name
1    Alice
2      Bob
3  Charlie

DataFrame 2:
   Age
2   25
3   30
4   22
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
Joined DataFrame (inner join):
      Name  Age
2      Bob   25
3  Charlie   30
Joined DataFrame (outer join):
      Name   Age
1    Alice   NaN
2      Bob  25.0
3  Charlie  30.0
4     None  22.0
Joined DataFrame (left join):
      Name   Age
2      Bob  25.0
3  Charlie  30.0
1    Alice   NaN
4     None  22.0
Joined DataFrame (right join):
      Name  Age
2      Bob   25
3  Charlie   30
4     None   22
```

```
# Performing an inner join on the indices
joined_df = df1.join(df2, how='inner')

# Displaying the joined DataFrame
print("Joined DataFrame (inner join):")
print(joined_df)

# Performing an outer join on the indices
joined_df = df1.join(df2, how='outer')

# Displaying the joined DataFrame
print("Joined DataFrame (outer join):")
print(joined_df)
```

```
Joined DataFrame (inner join):
      Name  Age
2      Bob   25
3  Charlie   30
Joined DataFrame (outer join):
      Name   Age
1    Alice   NaN
2      Bob  25.0
3  Charlie  30.0
4      NaN  22.0
```

```
# Performing an left join on the indices
joined_df = df1.join(df2, how='left')

# Displaying the joined DataFrame
print("Joined DataFrame (left join):")
print(joined_df)

# Performing an right join on the indices
joined_df = df1.join(df2, how='right')

# Displaying the joined DataFrame
print("Joined DataFrame (right join):")
print(joined_df)
```

```
Joined DataFrame (left join):
      Name   Age
1    Alice   NaN
2      Bob  25.0
3  Charlie  30.0
Joined DataFrame (right join):
      Name  Age
2      Bob   25
3  Charlie   30
4      NaN   22
```

8 #at example

```
1   #Pandas code
3   import pandas as pd
4
5   #Dataframe
6   df = pd.DataFrame({
7     'Name': ['Alice', 'Bob', 'Charlie'],
9     'Age': [25, 30, 22],
10  }, index=[0, 1, 2])
```

```
Original dataFrame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

print("Retrieved element: ", df.at[1, 'Name'])

```
Retrieved element:  Bob
```

Modifying a specific element
df.at[1, 'Name'] = 'Carter'

```
Modified dataFrame:
      Name  Age
0    Alice   25
1   Carter   30
2  Charlie   22
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
Retrieved element:  Bob

Modified dataFrame:
      Name  Age
0    Alice   25
1   Carter   30
2  Charlie   22
```

9 #iat example

```
1   #Pandas code
3   import pandas as pd
4
5   #Dataframe
6   df = pd.DataFrame({
7     'Name': ['Alice', 'Bob', 'Charlie'],
9     'Age': [25, 30, 22],
10  }, index=[0, 1, 2])
```

```
Original dataFrame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
```

print("Retrieved element: ", df.iat[1, 0])

```
Retrieved element:  Bob
```

Modifying a specific element
df.at[1, 0] = 'Carter'

```
Modified dataFrame:
      Name  Age
0    Alice   25
1   Carter   30
2  Charlie   22
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
Retrieved element:  Bob

Modified dataFrame:
      Name  Age
0    Alice   25
1   Carter   30
2  Charlie   22
```

10 #loc example

```
1    #Pandas code
3    import pandas as pd
4
5    #Dataframe
6    df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
7                      index=['cobra', 'viper', 'sidewinder'],
9                      columns=['max_speed', 'shield'])
10
```

```
Original dataFrame:
            max_speed  shield
cobra               1       2
viper               4       5
sidewinder          7       8
```

print("Slice with labels for row and single label for column:\n", df.loc['cobra':'viper', 'max_speed'])

```
Slice with labels for row and single label for column:
 cobra    1
viper     4
Name: max_speed, dtype: int64
```

print("Using boolean list:\n", df.loc[[False, False, True]])

```
Using boolean list:
             max_speed  shield
sidewinder           7       8
```

#FireDucks output with following cmdline:
python -mfireducks.imhook sample.py

```
Slice with labels for row and single label for column:
 cobra    1
viper    4
Name: max_speed, dtype: int64
Using boolean list:
             max_speed  shield
sidewinder           7       8
```

From the analysis above, it's evident that FireDucks executes code written for Pandas seamlessly, with matching results across various scenarios. Although a minor variance arises in cases of outer and right joins, where FireDucks outputs "None" instead of "NaN" in the column 'Name', its overall compatibility ensures a smooth transition for users. This level of alignment significantly minimizes the need for code adjustments, thereby reducing the learning curve when migrating from Pandas.

Here is a comparison of outputs from Pandas and Fireducks.

The Distinctive Offerings of FireDucks:

In addition to its compatibility efforts, FireDucks introduces a range of unique offerings that elevate its capabilities. Among these standout features are efficient memory handling and optimized execution models, setting FireDucks apart from its counterpart, Pandas.

List of Financial Potential, FireDucks in Action:

High-Frequency Trading (HFT)

In the fast-paced world of finance, High-Frequency Trading (HFT) stands out as a strategy that relies on lightning-fast decision-making and execution.

- FireDucks emerges as a valuable ally, equipped with blazing-fast filtering and aggregation capabilities that are ideal for analysing real-time market data feeds.
- One of FireDucks' key strengths lies in its ability to handle data processing tasks with remarkable speed and efficiency. Traders can leverage its powerful filtering and aggregation capabilities to extract valuable insights from complex datasets in real-time. This not only enables them to stay ahead of market movements but also allows for the rapid execution of trades with minimal latency.

Portfolio Optimization and Risk Management

At the heart of effective portfolio management lies the ability to swiftly analyze historical data and assess the associated risks

- FireDucks Quickly analyze vast amounts of historical price data and calculate risk metrics for various asset allocations.
- FireDucks facilitates efficient backtesting of various portfolio strategies, minimizing compute time and enhancing decision-making processes for investors and traders.

Fraud Detection

- Analyze large volumes of transaction data to identify anomalies that might indicate fraudulent activity.
- FireDucks' speed can enable real-time fraud detection thus helping to prevent financial losses

Algorithmic Trading Strategy Development

- Rapidly prototype and test various algorithmic trading strategies on historical data.
- FireDucks' fast iteration speed can help in quicker refinement and optimization of trading algorithms.

- **Quantitative Analysis (Quant) Workflows**
  - Perform complex calculations and feature engineering on financial data sets with ease.
  - Data manipulation tasks can be executed efficiently, allowing quants to focus on extracting insights and building robust models.

Conclusion

FireDucks offers a compelling option for data manipulation, particularly for users familiar with Pandas due to its API compatibility. However, it's crucial to go beyond just compatibility and evaluate its unique features, performance characteristics, and suitability for your specific needs. By understanding both its similarities and differences with Pandas, you can make an informed decision about whether FireDucks is the right tool for your data science projects.

FireDucks is a great option. It gives you the tools to work smarter and faster, which is why it's worth thinking about for your next project. It's not just another tool; it's one that makes your job easier and helps you do more.

References

FireDucks Documentation: https://github.com/fireducks-dev | FireDucks PyPI Page: https://pypi.org/project/fireducks/