CSCD01 Deliverable 3

Team Name: Waterboys

Rahmatullah Nikyar, Usman Siddiqui, Tony Zeng, Frank Xu, Weiqiang Zhang

Issue #31874:

Describing the Issue

When adding new columns to already aligned columns, the column type is defaulted at NaNs. However the issue poster believes it would be more of use to make new column creation dtypes the same as the previous columns dtypes. This would be useful as consistency is maintained and overall better for the user.

UML Links:

https://github.com/CSCD01/team_12-project/blob/master/d3/31874_changes.pdf

Code Traces:

```
Import pandas as pd
a = pd.DataFrame({“A”: [1,2], “B”: [pd.Timestamp(‘2000’), pd.NaT]})
b = pd.DataFrame({“A”: [1,2]})
a.align(b)[1].dtypes

A        int64
B      float64
dtype: object
```

Where it should be the case where the dtype of the new column aligned with A should maintain the same dtype. I.e B should have a dtype of datetime64[ns]. This doesn't happen because in the code

a.align(b)
-> frame.py class DataFrame method align line 3811 | delegates to NDFrame align
-> generic.py class NDFrame method align line 8409 | calls _align_frame after determining a and b are dataframes
-> generic.py class NDFrame method _align_frame line 8491 | calls _reindex_with_indexers on both frames a and b, passing in information about columns and indexes that need to be included
-> generic.py class NDFrame method _reindex_with_indexers line 4584 | iterates over each index supplied, and calls on the reindex_indexer on the dataframe’s blockmanager field for each non-None index
-> mangers.py class BlockManager method reindex_indexer line 1224 | delegates to _slice_take_blocks_ax0 after seeing that the alignment is on axis 0
-> mangers.py class BlockManager method _slice_take_blocks_ax0 line 1274 | iteratively calls _make_na_block to create new blocks for the new columns
-> mangers.py class BlockManager method _make_na_block line 1360 | returns a new block with the same type of the supplied fill value. If fill value is None, block is created as type float and filled with np.nan (numpy)

Description of how it could be fixed:
The reason why this issue exists is that from Nd.align if you go to the case where you're aligning a frame, the dtype of the other is lost and doesn't propagate down through the remaining methods. A fix for this will be to add an optional parameter to keep track of the dtype in the _reindex_ with _indexers, Block manager's reindex_indexer, Block manager slice_take_axis0 to Block Manager's _make_na_block. Then the others dtype will be in the parameter and we can replace the dtype of the new column being aligned with the originals instead of using Nan to fill it.

Issue #32450:

Describing the Issue
No current direct conversion between stringDtype series to Inte64Dtype series. The poster of the issue is able to convert to the latter type by indirectly calling the to_numeric method, but there are other issues when converting with the to_numeric method as well. To_numeric requires the unnessary parameter errors='coerce' and needs convert_dtypes to be called afterwards even though it should be returning Int64 Dtype directly. However, there should be a direct way to convert such types. The poster did make a mistake on their first test, where astype("Int64") was attempted, this is not supposed to work because Int64 isn't a dtype of numpy but int64 is. The problem lies in the fact that "astype('int64)" is called on a StringArray with panda's NA type, numpy does not know how to handle converting Series and Dataframe typings when it includes pandas.NA or numpy.nan.

UML Links:
https://github.com/CSCD01/team_12-project/blob/master/d3/32450_changes.pdf

Code Traces:
x = pd.Series(['1', pd.NA, '3'], dtype=pd.StringDtype())
x.astype('int64')

generic.py's astype method, line 5485 | called because there is a single dtype, being 'int64'
-> managers.py's astype method line 596 | calls self.apply function with "astype" parameter and dtype/copy/errors as **kwargs
-> managers.py's apply method line 383 | calls astype for the block, because it's a Series, there is only one block to loop through
-> blocks.py's astype, one of two paths can be taken 602 & 605, both will result in the same error | Because we initialized our Series with dtype=pd.StringDtype() parameter it is an extension and will go into 602 and call the value's astype with the new dtype
-> string_.py's astype method line 268 | checks if current dtype is the same as the requested one, just return the same thing or a copy of it if copy is requested, otherwise call the super astype
-> base.py's astype method line 448 | just creates a numpy array with the ndarray and new dtype.

This doesn’t work because numpy doesn’t handle dtype conversion with nan types in arrays.

Implementation Plan for 32450:
After digging through the codecase we discovered that using astype to turn an integer series into a string series will still work with a pandas.NA type in the series. The way pandas has handled the work around to make this work is having a mask object stored where it’s a list of booleans, having True values mirroring the pandas.NA type in the stored ndarray. This fixes the problem because the ndarray can just have integers in place of pandas.NA. IntegerArray inherits from BaseMaskedArray and StringArray just inherits from PandasArray. The high level plan to fix issue 32450 is going to be to make StringArray inherit from BaseMaskedArray and partly, if not fully implement StringArray. Since StringArray is used a lot everywhere in both Series and Dataframe we’re going to have to test thoroughly to make sure we don’t break any of the previous functionalities.

Select one and explain why we chose that one:
We chose the issue 32450 because of several factors. This specific issue has unexpected behaviors. Because Pandas is a large library, having consistent behavior is important to ensure reliability. An interesting aspect of the problem is that we get to examine an underlying Pandas data implementation problem. We want this opportunity to explore different low level components of Pandas and expand our knowledge. This improvement will also provide the users more options and lessen confusion. We have to look at multiple classes and really understand the underlying conversion that is going on and how the dtypes for Int and String really work.

Acceptance Tests for #32450:

```
x = pd.Series(['1', pd.NA, '3'], dtype=pd.StringDtype())
x.astype(“int64”)
0      1
1   <NA>
2      3
dtype: Int64
```

```
x = pd.DataFrame({‘a’: [‘1’, ‘3’], b: [pd.NA, ‘2’]}, dtype=pd.StringDtype())
x.astype(int)
       a      b
0      1   <NA>
1      3      2
```

Higher level hierarchy UML Design:
https://github.com/CSCD01/team_12-project/blob/master/d3/architectural_design.png

Architecture Document:

Modules:

IO - includes input/output tools for handling files. The most important module is api.py, where the majority of the imports for i/o is done.

core - Consists of data structures in the library itself, such as Series, DataFrame and PandasObject.

plotting - Extending the matplotlib to plot pandas objects.

Interesting Architectural Decisions:

- Pandas used a reuse-oriented design principle by heavily relying on numpy's array and data types for its underlying data for both DataFrame and Series objects.
    - To implement Pandas' own NA type that is type agnostic (which numpy arrays are not), they extended the numpy array to their own Pandas Arrays. Specifically, the pandas BaseMaskedArray class uses two numpy arrays, one to hold the data another called a 'mask' which is full of booleans. The boolean array signals which indexes of the main data array should be represented by the Pandas NA class.