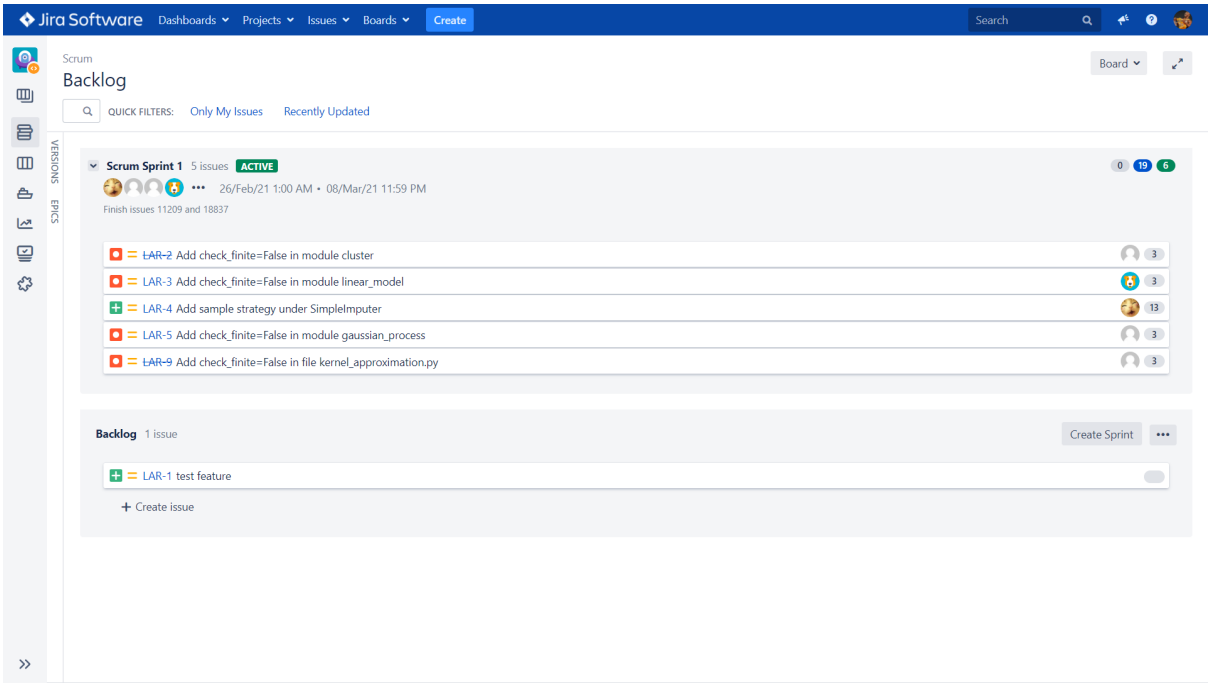
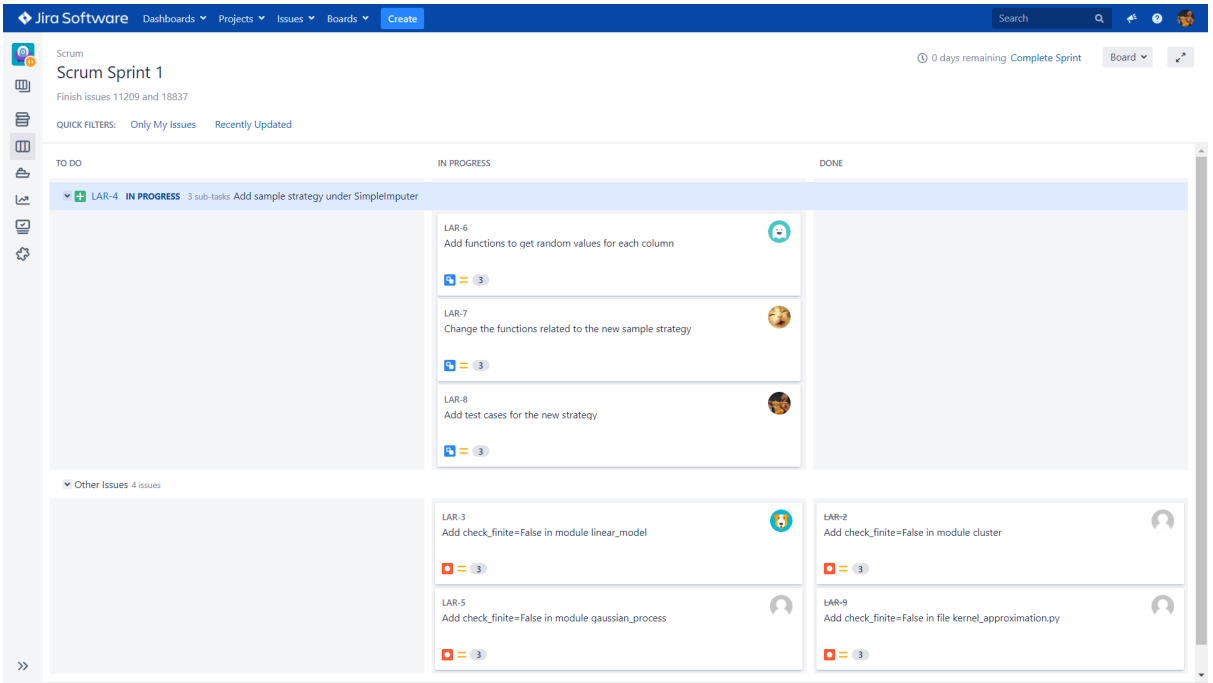
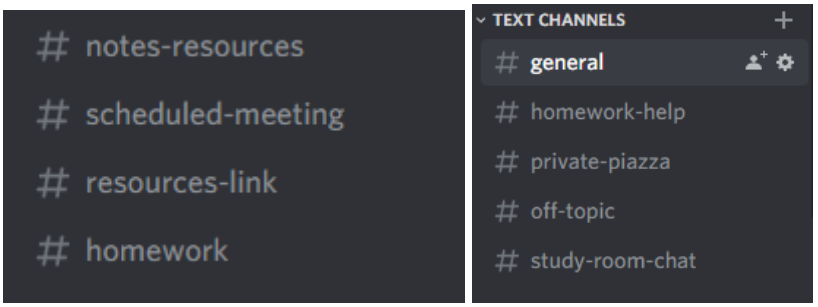
Project management:

1) We used Jira and Scrum. Below are two screenshots taken during the a2 sprint.

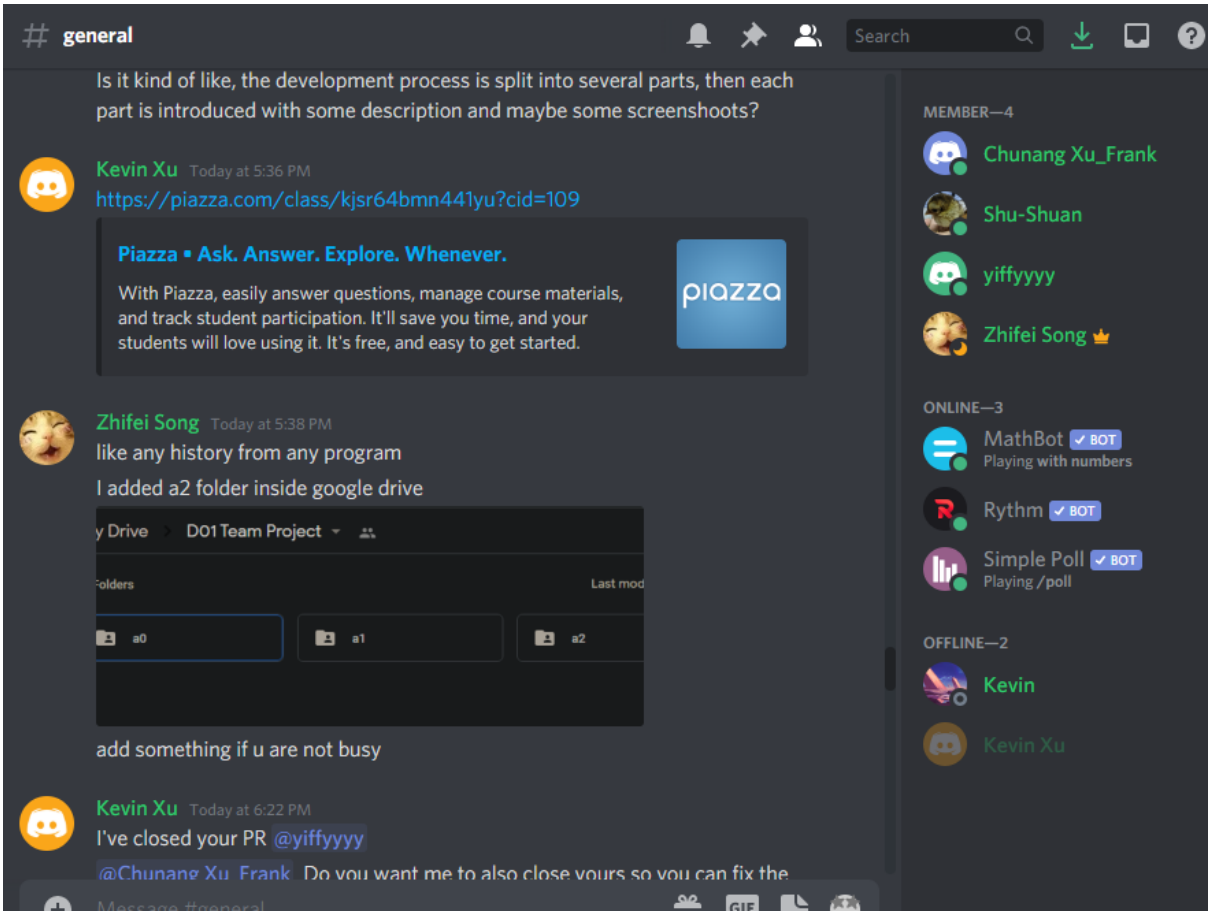




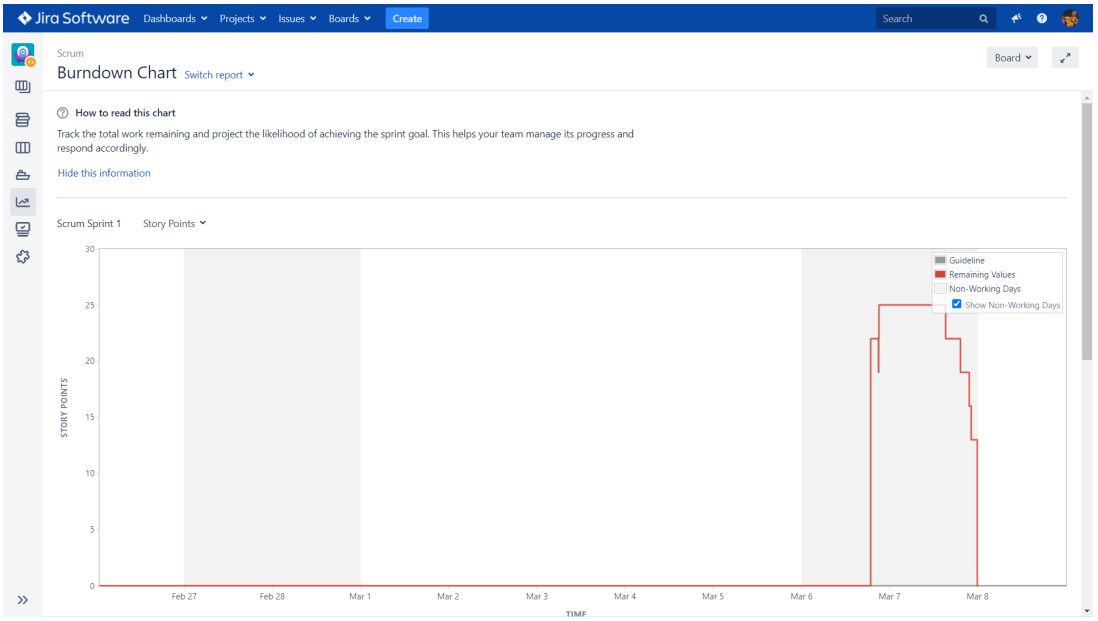
We mainly used discord to communicate, we organized our channels into different categories of discussion



We had scheduled meetings a few times a week to discuss ideas and work.



2) Burndown chart



3) Please see standup notes for meeting records in the same folder on the GitHub

4) What did you learn, what you could have done differently, what worked, what did not work [this may also be a summary for all stages if you like]

What we learned

- Communicate with team members. Created study rooms channel on Discord. Members can discuss and find instant help.
- Organize timeline to complete individual tasks
- Detail implementation of all strategies in SimpleImputer
- Detail implementation of _validate_data and _check_array
- learning how to breakdown and explore existing code, understand documentation and make good contributions
- planning and distributing work throughout
- On the first meeting of the sprint, each team member was assigned a part to understand. Then, each member introduced his/her assigned part to others so that everyone could have an overview of the issues quickly.

Could have done differently

- Committed to a project tracking software earlier, we had switched from discord channels to google docs folders, to Jira.
- Plan out stories with greater detail since features may have many more parts than initially assumed
- Increase time limits to allow for a change of plans, if one implementation seems too difficult

What worked well

- Good usage of the GitHub repo
- Good team communication on discord and effective meetings

For the duplicate check finite, you won't see visible differences, you can verify the changes by:

1) Searching scipy.linalg, sp.linalg or linalg (linalg might be imported differently)
2) Check if the methods for linalg functions take in check_finite=false (check_infite is True if not provided) and check_array or _validate_data is called above
3) Cd into each directory and run the tests

For a user to run the new Random Sample imputer you would use the following steps:

1) Import the required libraries numpy and simpleimputer
2)

```
import numpy as np
from sklearn.impute import SimpleImputer
```

3) Create an instance of the SimpleImputer, and set the strategy='sample'. A rand_state is a seed for the random sample generator and is optional

```
imp = SimpleImputer(missing_values=np.nan,strategy='sample',rand_state=0)
```

4) Fit the random sample imputer to a set of data using the imputer.fit function

```
imp.fit([[7, 1, 3], [4, np.nan, 6], [10, 1, 9]])
```

5) Use the transform function to impute values sampled at random from your fitted data

```
print(imp.transform([[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]))
```

Output will be consistent relative to the rand_state seed.

```
[[ 4.  2.  3.]
 [ 4.  1.  6.]
 [10.  1.  9.]]
```

To run the test cases, use pycharm and run the test folder for input in the sklear.impute.test. All tests should pass with green indicator.

Test folder:
https://github.com/UTSCCSCD01/course-project-large-software/tree/feature-LAR-4-sample-strategy/a2/scikit-learn/sklearn/impute/tests

Write a brief technical commentary on how your changes affect the design and/or the code of scikit-learn. List all relevant scikit-learn source code files that were added, modified or removed as part of your implementation work.

Bug #18837 Duplicate check_finite when calling scipy.linalg functions

In most scipy.linalg functions such as norm, svd, qr, eig, eigh, pinv, pinv2, … take *check_finite: bool* as a parameter which represents whether to check the input matrix contains only finite numbers. The default value of *check_finite* is set to *True*, and scikit-learn uses the default value when using these functions. We modified some scipy.linalg functions to take in *check_finite=False* in *kernel_approximation*, module *cluster*, *gaussian_process, and linear_module* because either *check_array()* or *_validate_data()* was called prior to the usage of the scipy.linalg functions. *check_array* checks if the input is a non-empty 2D array containing only finite values, and *_validate_data* calls *check_array* internally, so setting *check_finite=True* is redundant and significantly impacts the performance.
The ticket specifically mentioned no test modifications were needed. We believe this is because we only care about correctness and not performances in tests. Additionally, if both code and tests are modified at the same time, the tests might not capture any mistakes related to the code modification. We also can't test *check_finite* in scipy.linalg functions since it is an external library.

Feature# 11209 Add strategy="random" to SimpleImputer

Imputation for missing values can take multiple approaches. Some methods calculate the mean value of column vectors in an input space to replace missing values, and others use functions such as median, mode, or simply replace with a constant value. However, all of these methods fall short of accurately representing the variance in potential input data. The strategy="random sample" is a proposed way to remediate this problem; by sampling data at random via fitting an imputer to a set of data, one would be able to generate random

data that would more than likely fit missing values. The purpose would be to provide an unbiased, stochastic, and uniform distribution of data points over the imputed values, that would still maintain uniformity in its entirety.

The first implementation design chosen was to use a pseudorandom number generator for the basis of random sampling. This makes the imputed data random enough for any purposes but predictable so that it would be possible to re-create an imputation result more than once. Had we used a truly random generator, it would be impossible to replicate any imputed data given sufficient size.

The other strategy we decided to use was to sample randomly based on columns rather than the entire input dataset. This is to more accurately represent a column's values. In addition to that, for sparse data, n_zeros was considered. Sparse data in which missing values are 0 would force density on imputation, making it no-longer sparse. We aimed to maintain the sparsity of such data by sampling additional 0's as well.

For the fit implementation we had two ideas:

1) Store the entire fit dataset and on transform, pseudorandomly sample values
2) Store a list of unique values and probabilities, and use the python's pseudo-random random.choice function to sample values

We decided that the first implementation #1 would better suit our application, as the worst-case memory complexity of #2 is the same as the average memory complexity of #1, however #1 offers much faster fit and transform implementations, since #2's pseudo-random choice function requires high overhead processing, and additionally, to form the unique list, requires additional pre-processing.

In #1, pre-processing is limited to storing the transpose of the input dataset into ndarrays. Using NumPy's ndarray functions we are able to store the fit data into a usable form in only O(logN) iterations. Comparatively #2 would require iterating over the entire dataset and finding unique values, even with hashmaps would require O(N).

In transformation, #1 again comes out on top. Both implementations require looping over masked values to find which columns to omit from the output, however using random.choice incurs significant overhead to python's py.random library, whereas our implementation for #1 only requires a random_index value in the range of the size of the column vector. This way we use NumPy's pseudorandom generator to generate a number in O(1) time, then using contiguous arrays, we can index the random_index value in O(1) time again. Overall both implementations operate in O(N) however #1's coefficient is much lower, and therefore would be faster in practice.

List of files changed:

Files for Bug #18837

Changes made in sklearn/cluster:

https://github.com/UTSCCSCD01/course-project-large-software/commit/b5bef2c9df7d82680cc7ebeecbe0f7501ac8159c

Changes made in sklearn/kernel_approximation:

https://github.com/UTSCCSCD01/course-project-large-software/commit/a2f0912b31b565489aa0e193817d12400d6f0ed9

Changes made in sklearn/gaussian_process:

bug: LAR-5 added check_finite=false in _gpr.py · UTSCCSCD01/course-project-large-software@50a8dd8 (github.com)

bug: LAR-5 added check_finite=false in _gpc.py · UTSCCSCD01/course-project-large-software@ca53285 (github.com)

Changes made in sklearn/linear_model:

https://github.com/UTSCCSCD01/course-project-large-software/pull/5/commits/7ec15ba3342b06ee109bfb39444291b891e9661c

https://github.com/UTSCCSCD01/course-project-large-software/commit/2809af4626d442f3640aba1789986996e03dfec4

Files for Feature# 11209

Changes made in sklearn/impute/_base.py

https://github.com/UTSCCSCD01/course-project-large-software/blob/feature-LAR-4-sample-strategy/a2/scikit-learn/sklearn/impute/_base.py