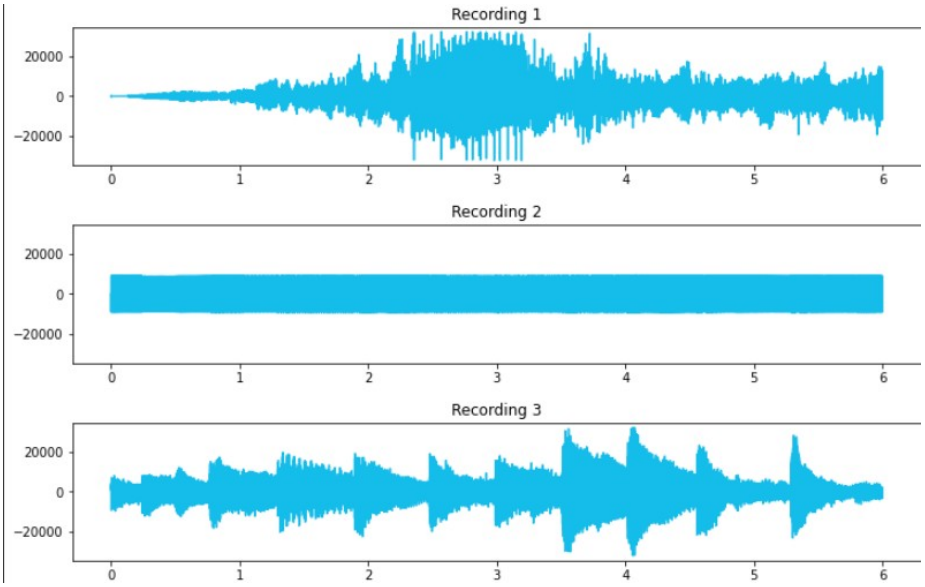
Lab 9
Abu Shahid
B20CS003

Question 1

Subpart 1: Read, Visualize and listen to Audio files

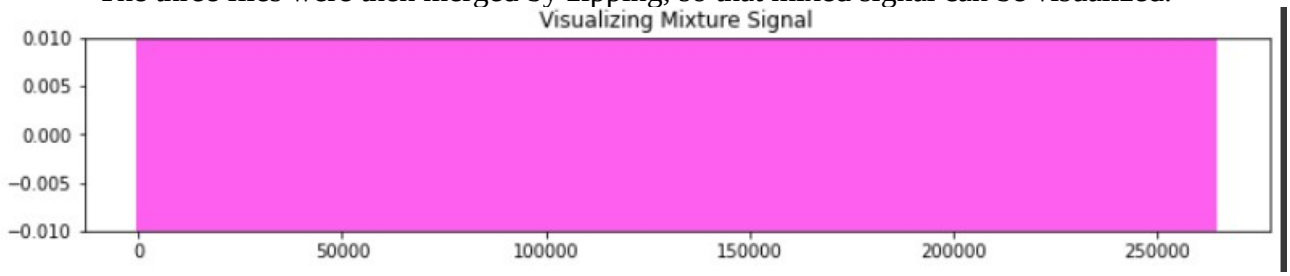
- Necessary libraies were imported.
- Data was read and processed as it is. No data cleaning attempt was made and crackling noise in signal one was taken as part of the data.
- Amplitude graphs of the pure input signals were then plotted.



- Audio files were analysed too.

Subpart 2: Extracting raw audio and merging the three files

- The three files were then merged by zipping, so that mixed signal can be visualized.



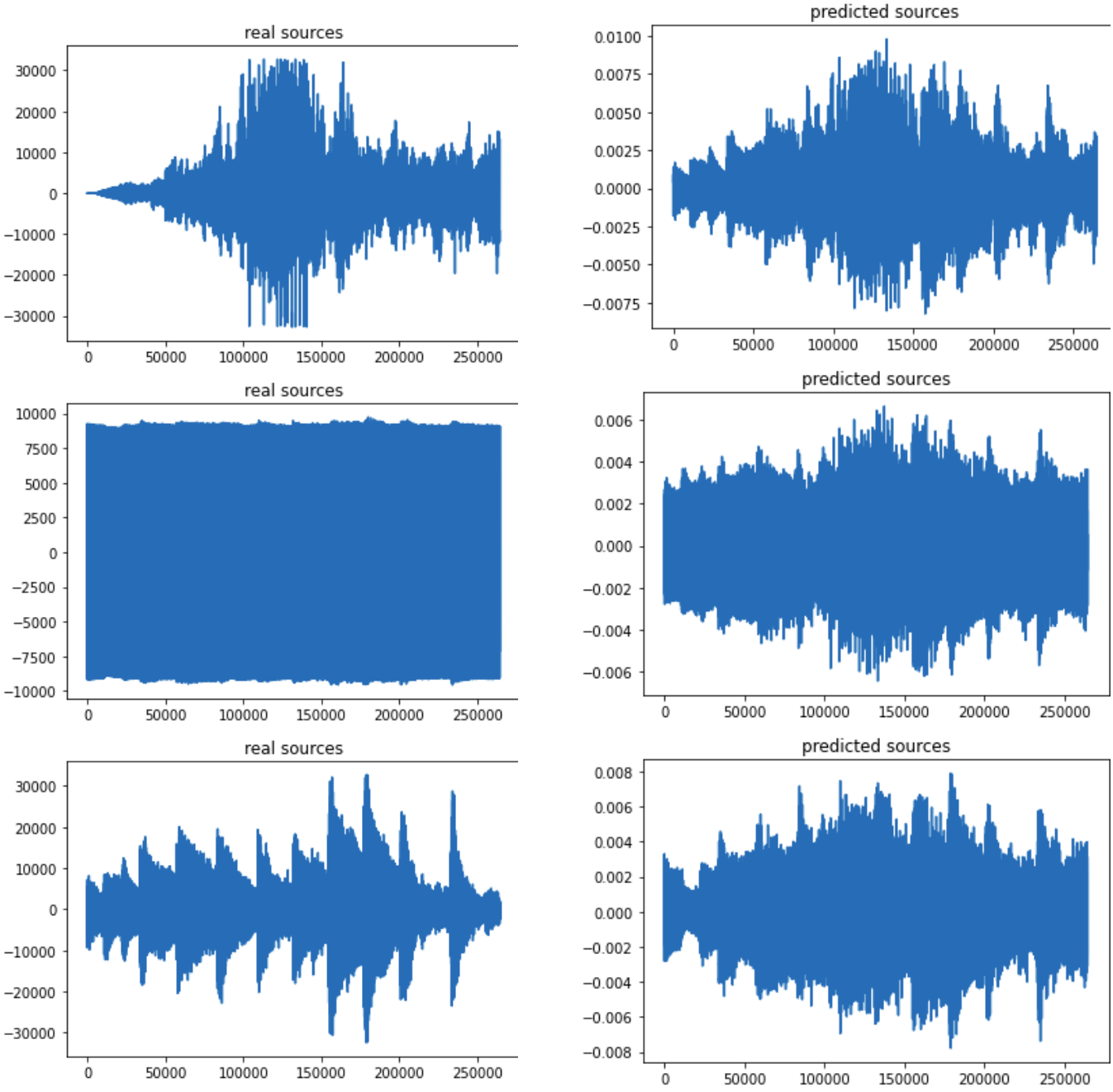
Subpart 3 and 4: Implementing ICA from scratch and reporting result

- To materialize this, we first centre the noisy merged niuse about the mean of the signal.
- Then the signal was ‘whitened’. That is an attempt was made to reduce the potential correlations between the 3 signals to possible 0, maintaining individual variance at 1. The covariance matrix of an such a whitened signal is expected to be identity.

$$\tilde{x} = ED^{-1/2}E^{T}x$$

- This was done as above, where D is the diagonal matrix of eigen values of our said covariance matrix and E is orthagonal matrix of eigenvectors.

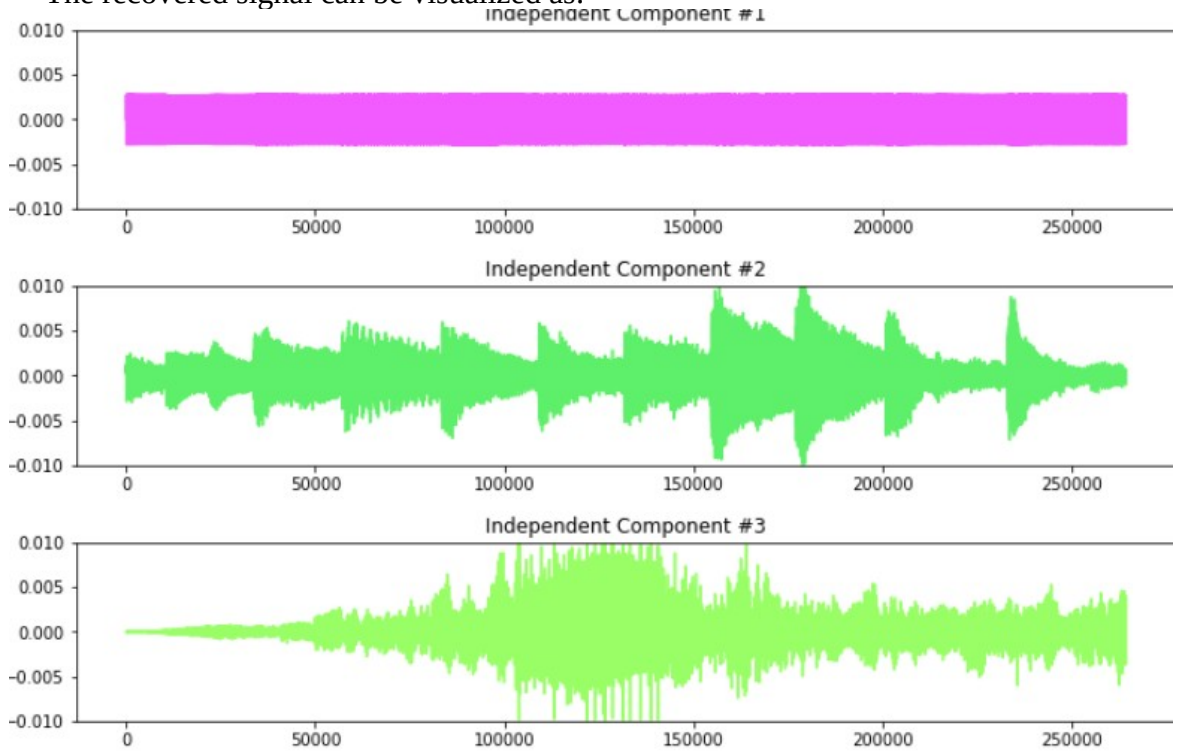
- Using this we calculate and update our ‘separation sieve’ matrix a said number of times or until a certain parameter is met, whichever earlier.
- Some parts of the implementation was referenced from: https://towardsdatascience.com/independent-component-analysis-ica-in-python-a0ef0db0955e
- Results of the same can be depicted as:



*for reasons, plot of predicted signal doesn’t convey accurate picture of the performance of the model.
**for this part an attempt was made to realize our signal into frequency domain from time doamin, using Discrete Fourier Transform and further proceed with analysis but results could not be materialized in time :(

Subpart 5 and 6: Implementing fast ICA and summarizing our results

- FastICA was imported from sklearn and noisy signal was fitted, with n_components set to 3
- The recovered signal can be visualized as:



which comes in very close agreement to our input signal.

- Even the audio files of the recovered signal seemed uncanny to the original.

Subpart 7: Difference between ICA and FastICA

- Refering to literature, the only advantage FastICA should give is about the reduced computational load leading to faster convergence in general. Since however our other variant from ICA is scratched, such a comparison doesn't have any grounds.
- Speaking of efficiency, FastICA was very resilient towards the noise and able to separate the three signals (two instruments and when white noise).
- Scratch implementtation of our ICA however was unable to somehow detect and filter out the second signal(white noise one).
- The first component was separated well-off(violin) however the next two components were a mixture(of piano and the white noise.)
- The reason for this is rooted in the assumption sustaining ICA algorithms.
  - The sources are statistically independent.
  - Each component has non-Gaussian distribution
  - and, the mixing system is determined.
- The fundamental restriction in ICA is independent components must be non-gaussian for ICA to be possible.
- The second (noise) signal is Gaussian as it has a normal distribution in time domain with an average time domain value being 0.

Question 2

Subpart 1: Data Preprocessing and Visualization

- Data was loaded and analysed.
- Categorical features were encoded and rows yielding to ‘nan’ values in any of the features were selectively dropped.
- This exercise was done for both train and test data.

Subpart 2: Creating SFS object

- Sequential Feature Selector was imported from mlxtend.feature_selection and object an SFS object was created using
  - DecisionTreeClassifier() as our predictor
  - number of features equal to 10
  - forward set to **True** and floating set as **False**
  - scoring metric kept as **accuracy**

```
clf = DecisionTreeClassifier(random_state=0)
sfs1 = SFS(clf,k_features=10, forward=True, floating=False,scoring='accuracy')
```

Subpart 3: Training and reporting

- X_train and y_train were fitted.
- The average validation score (cv=5) was reported to be 95.06%
- Top 10 features were:
  - `'Customer Type', 'Type of Travel', 'Class', 'Inflight wifi service', 'Gate location', 'Online boarding', 'Seat comfort', 'Inflight entertainment', 'Baggage handling', 'Inflight service'`

```
best accuracy achieved: 0.95066317381588 59
```

Subpart 4: Toggling

Implementing SBS from scratch:

- SBS was implemented from scratch with the following description
  sbs_scratch(...)
- Parameter list:
  - model: model to be used for feature selection
  - X: training data
  - y: training labels
  - features: number of features to be selected
- The function trains the data on all the features initially. Then it removes one feature at a time and calculates the accuracy. The feature whose dropping gives the best result is removed for good.
- This exercise is followed till the number of remaining features is equal to ***features***
- The function finally returns a list which packs the average validation score with the best feature set and names of the said features:
- The accuracy of our implementation was reported to be: 93.46%
- and best feature set came out to be: ' Customer Type', 'Type of Travel', 'Inflight wifi service', 'Gate location', 'Online boarding', 'Seat comfort', 'Inflight entertainment', 'On-board service', 'Baggage handling', 'Inflight service'
- The feature list returned is same as given by the inbuilt implementation previously.

```
accuracy of SBS scratch is:  0.9346
```

- Further for this subpart, objects for **sfs, sbs, sffs and sbfs** were created, with classifier as Decision Tree and number of folds for cross validation set equal to 4.
- There performances can be summarized as:

```
for SFS
best accuracy achieved: 0.9499198910655715
 cv scores are:  [0.94883972 0.94980501 0.94976446 0.95127037]


for SBS
best accuracy achieved: 0.9301
 cv scores are:  [0.9264 0.9312 0.9368 0.926 ]


for SFFS
best accuracy achieved: 0.9391
cv scores are:  [0.934  0.9428 0.9424 0.9372]


for SBFS
best accuracy achieved: 0.9387
 cv scores are:  [0.938  0.9424 0.942  0.9324]
```

Subpart 5: Visualizing the output from feature selection for all 4 congigurations

sfs_pd

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(13,)	[0.7897602224024094, 0.7918066334607514, 0.793...	0.790335	(Online boarding,)	0.004065	0.002536	0.001464
2	(5, 13)	[0.8483339125062743, 0.8511911656820726, 0.850...	0.849615	(Type of Travel, Online boarding)	0.002085	0.0013	0.000751
3	(5, 8, 13)	[0.8914629908490675, 0.8920421637901077, 0.892...	0.891249	(Type of Travel, Inflight wifi service, Online...	0.002214	0.001381	0.000797
4	(5, 8, 11, 13)	[0.9192246804895942, 0.9228927757828488, 0.922...	0.921733	(Type of Travel, Inflight wifi service, Gate l...	0.002346	0.001463	0.000845
5	(3, 5, 8, 11, 13)	[0.9277578284875864, 0.9284914475462374, 0.929...	0.928828	(Customer Type, Type of Travel, Inflight wifi ...	0.001202	0.00075	0.000433
6	(3, 5, 8, 11, 13, 18)	[0.9393412873083903, 0.9425846557782154, 0.939...	0.9413	(Customer Type, Type of Travel, Inflight wifi ...	0.002925	0.001825	0.001053
7	(3, 5, 6, 8, 11, 13, 18)	[0.9462913626008727, 0.9488011120120468, 0.948...	0.948269	(Customer Type, Type of Travel, Class, Infligh...	0.001897	0.001183	0.000683
8	(3, 5, 6, 8, 11, 13, 18, 20)	[0.9495347310706977, 0.9503455731881539, 0.950...	0.950615	(Customer Type, Type of Travel, Class, Infligh...	0.001315	0.000821	0.000474
9	(3, 5, 6, 8, 11, 13, 14, 18, 20)	[0.9488397235414495, 0.9508089115409861, 0.951...	0.95076	(Customer Type, Type of Travel, Class, Infligh...	0.00193	0.001204	0.000695
10	(3, 5, 6, 8, 11, 13, 14, 15, 18, 20)	[0.9488397235414495, 0.9498050117765164, 0.949...	0.94992	(Customer Type, Type of Travel, Class, Infligh...	0.001395	0.00087	0.000502

sffs_pd

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(13,)	[0.7788, 0.7864, 0.7816, 0.7892]	0.784	(13,)	0.006492	0.00405	0.002338
2	(5, 13)	[0.8444, 0.8456, 0.844, 0.8512]	0.8463	(5, 13)	0.004632	0.00289	0.001668
3	(5, 8, 13)	[0.888, 0.8936, 0.8868, 0.8912]	0.8899	(5, 8, 13)	0.004286	0.002674	0.001544
4	(5, 8, 13, 18)	[0.912, 0.9144, 0.9152, 0.9252]	0.9167	(5, 8, 13, 18)	0.00809	0.005047	0.002914
5	(3, 5, 8, 13, 18)	[0.9272, 0.9336, 0.936, 0.9332]	0.9325	(3, 5, 8, 13, 18)	0.005197	0.003242	0.001872
6	(3, 5, 8, 13, 14, 18)	[0.9332, 0.9316, 0.942, 0.9332]	0.935	(3, 5, 8, 13, 14, 18)	0.006562	0.004094	0.002364
7	(3, 5, 6, 8, 13, 14, 18)	[0.9352, 0.9296, 0.9404, 0.9332]	0.9346	(3, 5, 6, 8, 13, 14, 18)	0.006258	0.003904	0.002254
8	(3, 5, 6, 8, 13, 14, 18, 20)	[0.9356, 0.9372, 0.944, 0.932]	0.9372	(3, 5, 6, 8, 13, 14, 18, 20)	0.00698	0.004354	0.002514
9	(3, 5, 6, 8, 13, 14, 15, 18, 20)	[0.9356, 0.9416, 0.9412, 0.934]	0.9381	(3, 5, 6, 8, 13, 14, 15, 18, 20)	0.005372	0.003351	0.001935
10	(3, 5, 6, 8, 11, 13, 14, 15, 18, 20)	[0.934, 0.9428, 0.9424, 0.9372]	0.9391	(3, 5, 6, 8, 11, 13, 14, 15, 18, 20)	0.005901	0.003681	0.002125

sbfs_pd

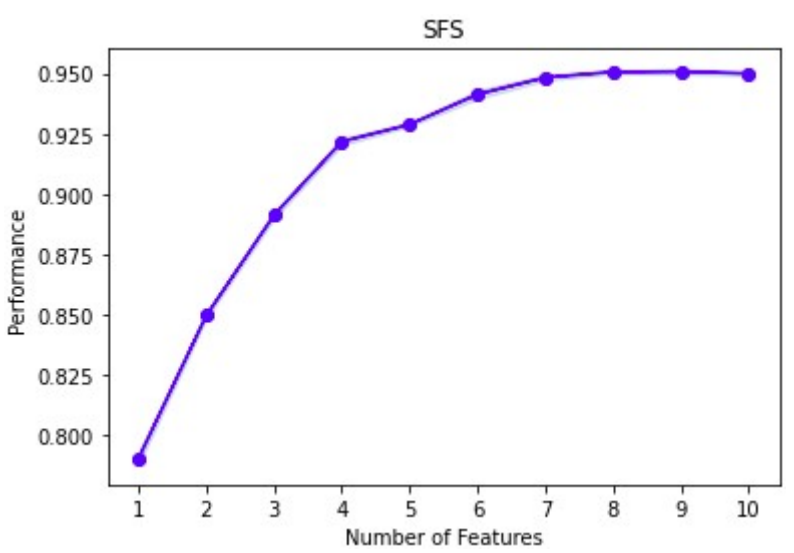
	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
24	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.9148, 0.9264, 0.9312, 0.8868]	0.9148	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	0.02762	0.01723	0.009948
23	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	[0.914, 0.9196, 0.9208, 0.928]	0.9206	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0.007989	0.004984	0.002877
22	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	[0.9156, 0.9304, 0.9308, 0.9252]	0.9255	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	0.009823	0.006128	0.003538
21	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	[0.9228, 0.9296, 0.93, 0.9244]	0.9267	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	0.005056	0.003154	0.001821
20	(1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, ...	[0.9204, 0.932, 0.9296, 0.9316]	0.9284	(1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, ...	0.007546	0.004707	0.002718
19	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9256, 0.9256, 0.9292, 0.9308]	0.9278	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.003641	0.002272	0.001311
18	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9196, 0.9256, 0.9344, 0.9344]	0.9285	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.01005	0.00627	0.00362
17	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9244, 0.928, 0.9312, 0.9304]	0.9285	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.004238	0.002644	0.001526
16	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9236, 0.9304, 0.9312, 0.9252]	0.9276	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.005229	0.003262	0.001883
15	(3, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 18, 1...	[0.9304, 0.9344, 0.9372, 0.928]	0.9325	(3, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 18, 1...	0.005688	0.003548	0.002049
14	(3, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 18, 1...	[0.9288, 0.9328, 0.9348, 0.936]	0.9331	(3, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 18, 1...	0.004381	0.002733	0.001578
13	(3, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20)	[0.9332, 0.93, 0.936, 0.9304]	0.9324	(3, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20)	0.003874	0.002417	0.001395
12	(3, 5, 8, 10, 12, 13, 14, 15, 18, 19, 20, 21)	[0.9352, 0.9352, 0.94, 0.9308]	0.9353	(3, 5, 8, 10, 12, 13, 14, 15, 18, 19, 20, 21)	0.005216	0.003254	0.001879
11	(3, 5, 8, 10, 12, 13, 14, 15, 18, 20, 21)	[0.9384, 0.9392, 0.9396, 0.9284]	0.9364	(3, 5, 8, 10, 12, 13, 14, 15, 18, 20, 21)	0.007436	0.004639	0.002678
10	(3, 5, 8, 10, 13, 14, 15, 18, 20, 21)	[0.938, 0.9424, 0.942, 0.9324]	0.9387	(3, 5, 8, 10, 13, 14, 15, 18, 20, 21)	0.00645	0.004024	0.002323

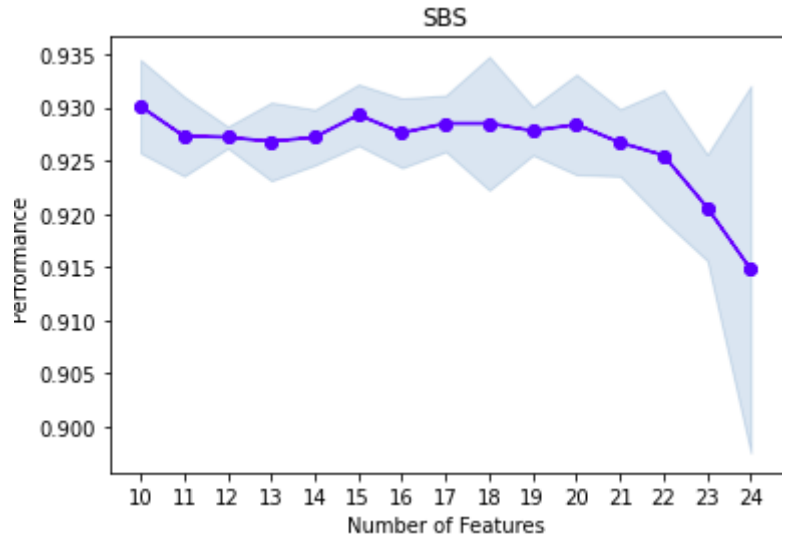
sbs_pd

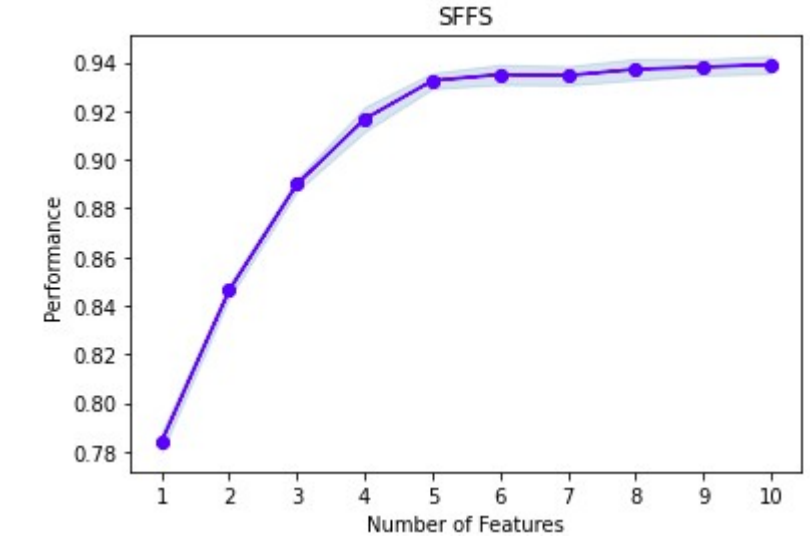
	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
24	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.9148, 0.9264, 0.9312, 0.8868]	0.9148	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	0.02762	0.01723	0.009948
23	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	[0.914, 0.9196, 0.9208, 0.928]	0.9206	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0.007989	0.004984	0.002877
22	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	[0.9156, 0.9304, 0.9308, 0.9252]	0.9255	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	0.009823	0.006128	0.003538
21	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	[0.9228, 0.9296, 0.93, 0.9244]	0.9267	(1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 1...	0.005056	0.003154	0.001821
20	(1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, ...	[0.9204, 0.932, 0.9296, 0.9316]	0.9284	(1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, ...	0.007546	0.004707	0.002718
19	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9256, 0.9256, 0.9292, 0.9308]	0.9278	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.003641	0.002272	0.001311
18	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9196, 0.9256, 0.9344, 0.9344]	0.9285	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.01005	0.00627	0.00362
17	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9244, 0.928, 0.9312, 0.9304]	0.9285	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.004238	0.002644	0.001526
16	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.9236, 0.9304, 0.9312, 0.9252]	0.9276	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.005229	0.003262	0.001883
15	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	[0.93, 0.9328, 0.9296, 0.9248]	0.9293	(1, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16,...	0.00461	0.002876	0.00166
14	(1, 3, 4, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18...	[0.924, 0.9284, 0.9308, 0.9256]	0.9272	(1, 3, 4, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18...	0.00418	0.002608	0.001506
13	(1, 3, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18, 19)	[0.9268, 0.9216, 0.932, 0.9268]	0.9268	(1, 3, 5, 8, 10, 11, 12, 13, 14, 15, 16, 18, 19)	0.005894	0.003677	0.002123
12	(1, 3, 5, 8, 11, 12, 13, 14, 15, 16, 18, 19)	[0.9284, 0.9256, 0.9276, 0.9272]	0.9272	(1, 3, 5, 8, 11, 12, 13, 14, 15, 16, 18, 19)	0.001635	0.00102	0.000589
11	(1, 3, 5, 8, 11, 12, 13, 14, 15, 16, 18)	[0.93, 0.9216, 0.9312, 0.9264]	0.9273	(1, 3, 5, 8, 11, 12, 13, 14, 15, 16, 18)	0.005987	0.003735	0.002156
10	(3, 5, 8, 11, 12, 13, 14, 15, 16, 18)	[0.9264, 0.9312, 0.9368, 0.926]	0.9301	(3, 5, 8, 11, 12, 13, 14, 15, 16, 18)	0.007015	0.004376	0.002527

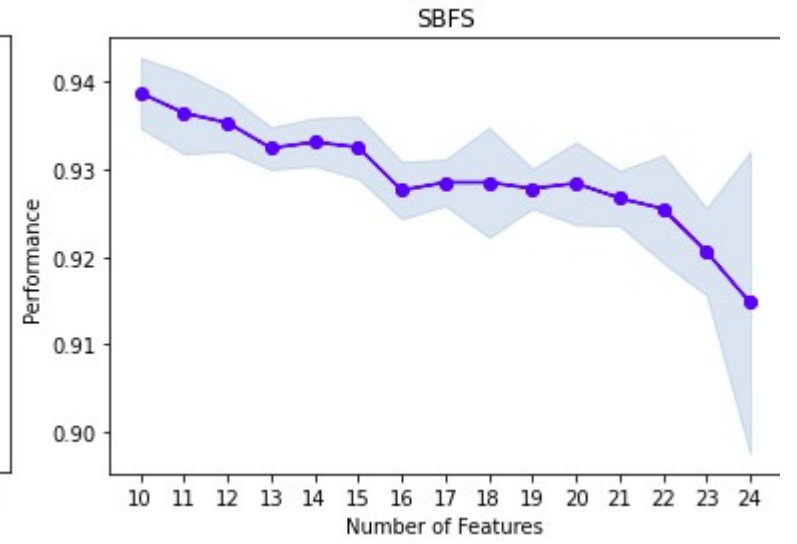
Subpart 6: Plotting

- For this part, performance was plotted against the number of features and same has been attached below for all the configurations.





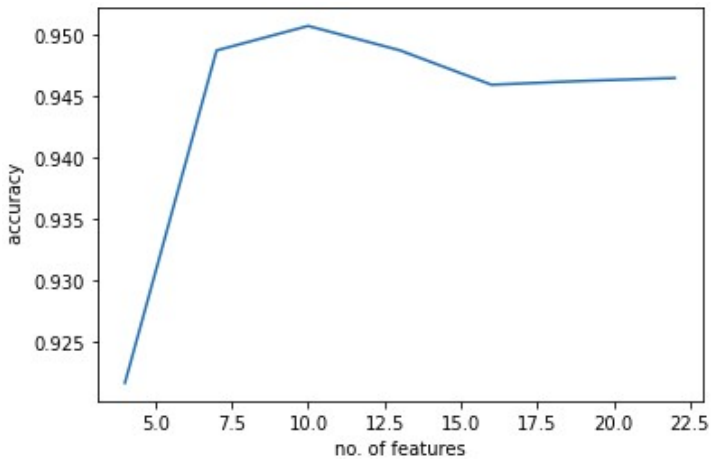




Subpart 7: Varying number of features and reporting the performance

- Number of features was varied from 4 to 24 with steps of three and the resultant accuracy can be reported as:

```
best accuracy achieved with 4 features is: 0.9217136206489922
best accuracy achieved with 7 features is: 0.9486746417539573
best accuracy achieved with 10 features is: 0.9506631738158859
best accuracy achieved with 13 features is: 0.9486842942635649
best accuracy achieved with 16 features is: 0.945894566164515
best accuracy achieved with 19 features is: 0.9462034469378782
best accuracy achieved with 22 features is: 0.9464351276690346
```



- With a unit step size, same exercise was repeated with number of features centred around 10 and the maxima in performance was indeed at number of features equal to 10.