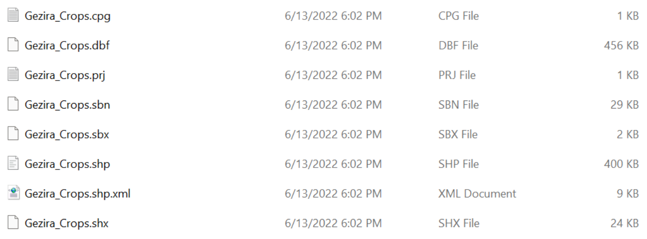
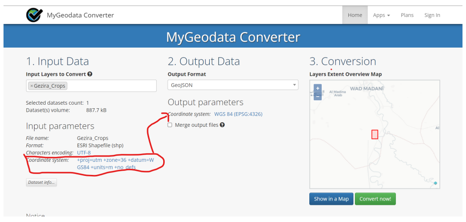
1. Abstract: (L)
2. Introduction (H): problem, assumption, limitations
3. Objective: (targets, milestones)
4. Related work: (M)
    - literature review.
    - Data Source.
    - Model Architecture.
5. Dataset and features engineering (H): (bottlenecks, libraries/tech used)
    - Initial data format.
        - The data is shared from our sources in a SHAPEFILE format.
        - The shapefile format is a geospatial vector data format for a geographic information system (GIS)
        - A shapefile often has information that identifies which coordinate system was used to define its features (WGS84, UTM)



Gezira_Crops.cpg	6/13/2022 6:02 PM	CPG File	1 KB
Gezira_Crops.dbf	6/13/2022 6:02 PM	DBF File	456 KB
Gezira_Crops.prj	6/13/2022 6:02 PM	PRJ File	1 KB
Gezira_Crops.sbn	6/13/2022 6:02 PM	SBN File	29 KB
Gezira_Crops.sbx	6/13/2022 6:02 PM	SBX File	2 KB
Gezira_Crops.shp	6/13/2022 6:02 PM	SHP File	400 KB
Gezira_Crops.shp.xml	6/13/2022 6:02 PM	XML Document	9 KB
Gezira_Crops.shx	6/13/2022 6:02 PM	SHX File	24 KB

        - This data is then converted by online tools to GEOJSON format which is easier to handle and more compact the shapefiles.
        - GeoJSON is an open standard geospatial data interchange format that represents simple geographic features and their nonspatial attributes. Based on JavaScript Object Notation (JSON), GeoJSON is a format for encoding a variety of geographic data structures.
        - The Final step is converting the coordinates to the WGS84 Latitude Longitude referencing system.
        - 

MyGeodata Converter

1. Input Data

Input Layers to Convert: Gezira_Crops

Selected datasets count: 1
Dataset(s) volume: 887.7 kB

Input parameters

File name: Gezira_Crops
Format: ESRI Shapefile (shp)
Characters encoding: UTF-8
Coordinate system: +proj=utm +zone=36 +datum=WGS84 +units=m +no_defs

2. Output Data

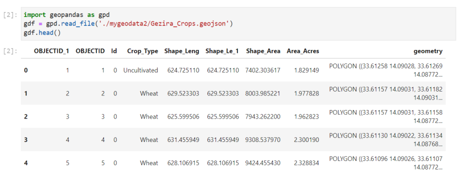
Output Format: GeoJSON

Output parameters

Coordinate system: WGS 84 (EPSG:4326)
Merge output files

3. Conversion

Layers Extent Overview Map

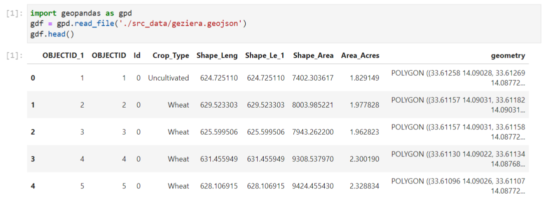
        - 

```
[2]: import geopandas as gpd
     gdf = gpd.read_file('./mygeodata2/Gezira_Crops.geojson')
     gdf.head()
```

[2]:

	OBJECTID_1	OBJECTID	Id	Crop_Type	Shape_Leng	Shape_Le_1	Shape_Area	Area_Acres	geometry
0	1	1	0	Uncultivated	624.725110	624.725110	7402.303617	1.829149	POLYGON ((33.61258 14.09028, 33.61269 14.08772...
1	2	2	0	Wheat	629.523303	629.523303	8003.985221	1.977828	POLYGON ((33.61157 14.09031, 33.61182 14.09031...
2	3	3	0	Wheat	625.599506	625.599506	7943.262200	1.962823	POLYGON ((33.61157 14.09031, 33.61158 14.08772...
3	4	4	0	Wheat	631.455949	631.455949	9308.537970	2.300190	POLYGON ((33.61130 14.09022, 33.61134 14.08768...
4	5	5	0	Wheat	628.106915	628.106915	9424.455430	2.328834	POLYGON ((33.61096 14.09026, 33.61107 14.08772...

        - These GeoJson files are then stored as ‘Raw Data’

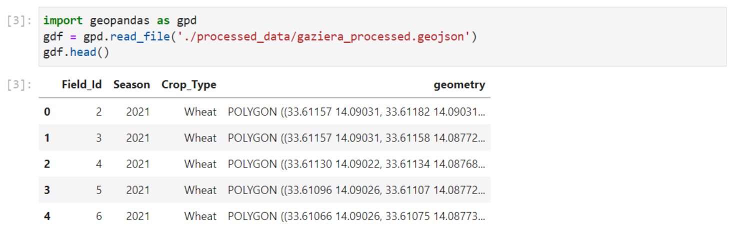


```
[1]: import geopandas as gpd
     gdf = gpd.read_file('./src_data/geziera.geojson')
     gdf.head()
```

[1]:

	OBJECTID_1	OBJECTID	Id	Crop_Type	Shape_Leng	Shape_Le_1	Shape_Area	Area_Acres	geometry
0	1	1	0	Uncultivated	624.725110	624.725110	7402.303617	1.829149	POLYGON ((33.61258 14.09028, 33.61269 14.08772...
1	2	2	0	Wheat	629.523303	629.523303	8003.985221	1.977828	POLYGON ((33.61157 14.09031, 33.61182 14.09031...
2	3	3	0	Wheat	625.599506	625.599506	7943.262200	1.962823	POLYGON ((33.61157 14.09031, 33.61158 14.08772...
3	4	4	0	Wheat	631.455949	631.455949	9308.537970	2.300190	POLYGON ((33.61130 14.09022, 33.61134 14.08768...
4	5	5	0	Wheat	628.106915	628.106915	9424.455430	2.328834	POLYGON ((33.61096 14.09026, 33.61107 14.08772...

    - Finally, these columns are filtered(unwanted crops are removed), renamed, and recalculated so that all the sources (Qadarif, Managil, Gaziera) have the same structure as shown below:

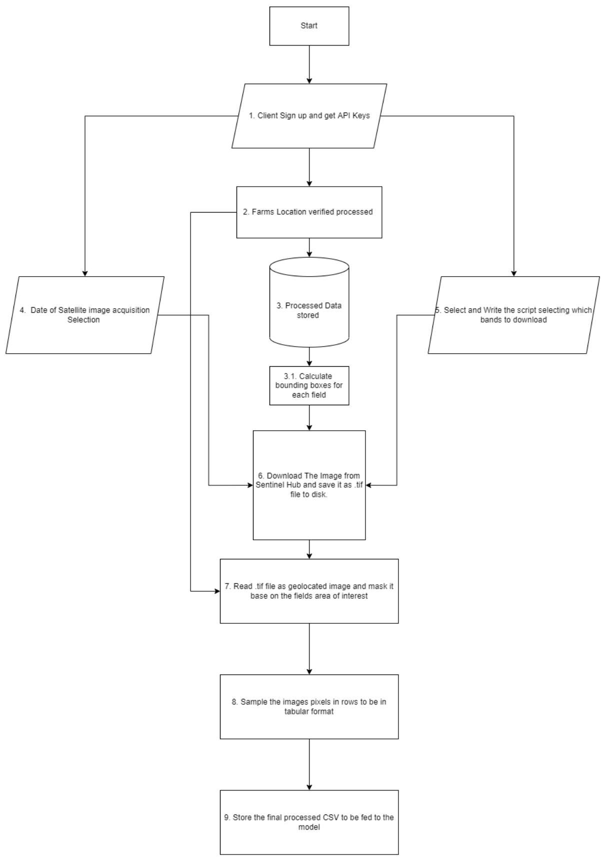


```
[3]: import geopandas as gpd
     gdf = gpd.read_file('./processed_data/gaziera_processed.geojson')
     gdf.head()
```

[3]:

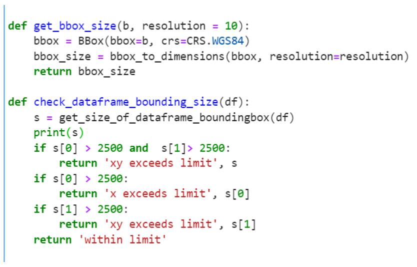
	Field_Id	Season	Crop_Type	geometry
0	2	2021	Wheat	POLYGON ((33.61157 14.09031, 33.61182 14.09031...
1	3	2021	Wheat	POLYGON ((33.61157 14.09031, 33.61158 14.08772...
2	4	2021	Wheat	POLYGON ((33.61130 14.09022, 33.61134 14.08768...
3	5	2021	Wheat	POLYGON ((33.61096 14.09026, 33.61107 14.08772...
4	6	2021	Wheat	POLYGON ((33.61066 14.09026, 33.61075 14.08773...

    -
    - These are then stored in the processed_data Folder
- Data extraction from sentinel hub.



Start
1. Client Sign up and get API Keys
2. Farms Location verified processed
3. Processed Data stored
3.1. Calculate bounding boxes for each field
4. Date of Satellite image acquisition Selection
5. Select and Write the script selecting which bands to download
6. Download The Image from Sentinel Hub and save it as .tif file to disk.
7. Read .tif file as geolocated image and mask it base on the fields area of interest
8. Sample the images pixels in rows to be in tabular format
9. Store the final processed CSV to be fed to the model

1. sign up and get API Keys from https://sentinelhub-py.readthedocs.io/en/latest/configure.html
2. Check the size of each farm and make sure it does not exceed the Sentinelhub Image size limit



```
def get_bbox_size(b, resolution = 10):
    bbox = BBox(bbox=b, crs=CRS.WGS84)
    bbox_size = bbox_to_dimensions(bbox, resolution=resolution)
    return bbox_size

def check_dataframe_bounding_size(df):
    s = get_size_of_dataframe_boundingbox(df)
    print(s)
    if s[0] > 2500 and  s[1]> 2500:
        return 'xy exceeds limit', s
    if s[0] > 2500:
        return 'x exceeds limit', s[0]
    if s[1] > 2500:
        return 'xy exceeds limit', s[1]
    return 'within limit'
```

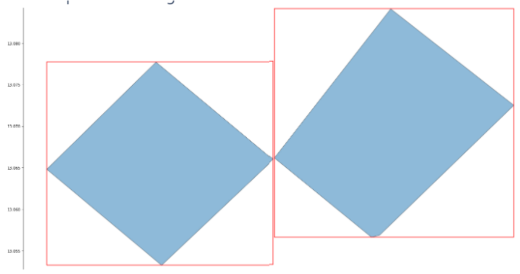
3. -

4. The Dates to acquire images are selected based on:
    a. The season in which the crops are planted and harvested(June-December).
    b. The frequency of the desired acquisition (One-Shot Per Month)

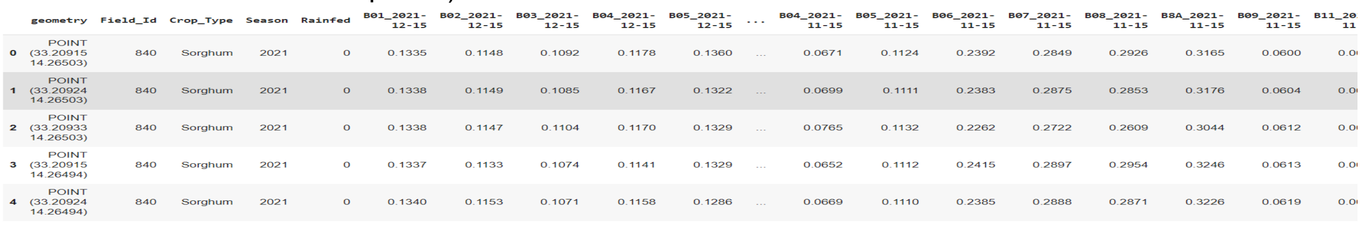
5. We use an eval script to describe the types of bands and their resolution and extra calculations “sentinel-2 carry a multispectral imager with a swath of 290 km. The imager provides a versatile set of 13 spectral bands spanning from the visible and near-infrared to the shortwave infrared, featuring four spectral bands at 10 m, six bands at 20 m, and three bands at 60 m spatial resolution. As indices primarily deal with combining various band reflectances, the table of 13 bands is given here for reference (see here for details). The Sentinel-2 bands at your disposal are *B01*, *B02*, *B03*, *B04*, *B05*, *B06*, *B07*, *B08*, *B8A*, *B09*, *B10*, *B11,* and *B12*.”
6. Use The SenHub API to request the data as in the example below:

```
request_true_color = SentinelHubRequest(
evalscript=evalscript_true_color,
input_data=[
SentinelHubRequest.input_data(
       data_collection=DataCollection.SENTINEL2_L1C,
       time_interval=("2020-06-12", "2020-06-13"),)],
responses=[SentinelHubRequest.output_response("default", MimeType.PNG)],
bbox=betsiboka_bbox,
size=betsiboka_size,
config=config,)
true_color_imgs = request_true_color.get_data()
```

7. Mask the image to only fit the area of interest:



8. Convert the Images to tabular format with pixel values(lat, long points) as rows:

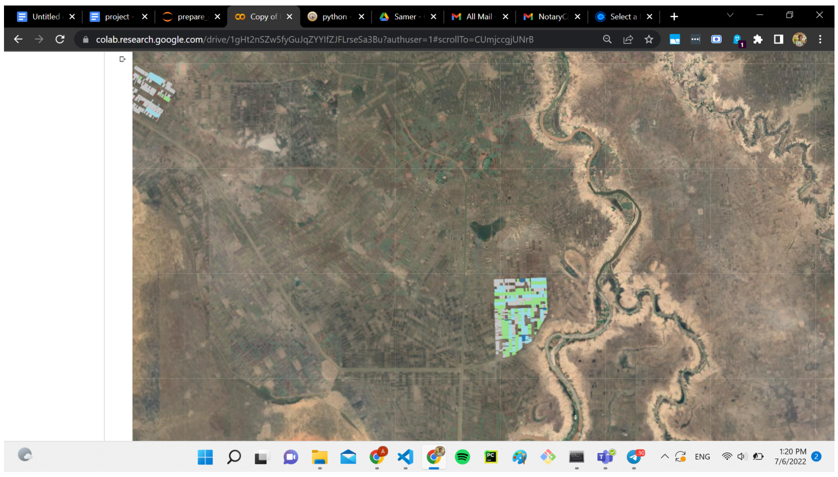


	geometry	Field_Id	Crop_Type	Season	Rainfed	B01_2021-12-15	B02_2021-12-15	B03_2021-12-15	B04_2021-12-15	B05_2021-12-15	...	B04_2021-11-15	B05_2021-11-15	B06_2021-11-15	B07_2021-11-15	B08_2021-11-15	B8A_2021-11-15	B09_2021-11-15	B11_20 11
0	POINT (33.20915 14.26503)	840	Sorghum	2021	0	0.1335	0.1148	0.1092	0.1178	0.1360	...	0.0671	0.1124	0.2392	0.2849	0.2926	0.3165	0.0600	0.0
1	POINT (33.20924 14.26503)	840	Sorghum	2021	0	0.1338	0.1149	0.1085	0.1167	0.1322	...	0.0699	0.1111	0.2383	0.2875	0.2853	0.3176	0.0604	0.0
2	POINT (33.20933 14.26503)	840	Sorghum	2021	0	0.1338	0.1147	0.1104	0.1170	0.1329	...	0.0765	0.1132	0.2262	0.2722	0.2609	0.3044	0.0612	0.0
3	POINT (33.20915 14.26494)	840	Sorghum	2021	0	0.1337	0.1133	0.1074	0.1141	0.1329	...	0.0652	0.1112	0.2415	0.2897	0.2954	0.3246	0.0613	0.0
4	POINT (33.20924 14.26494)	840	Sorghum	2021	0	0.1340	0.1153	0.1071	0.1158	0.1286	...	0.0669	0.1110	0.2385	0.2888	0.2871	0.3226	0.0619	0.0

9. These CSV Files are finally saved as separate three files(Gaziera.csv, Mangil.csv & Qadarif.csv) and uploaded to shared storage.

- Feature engineering.

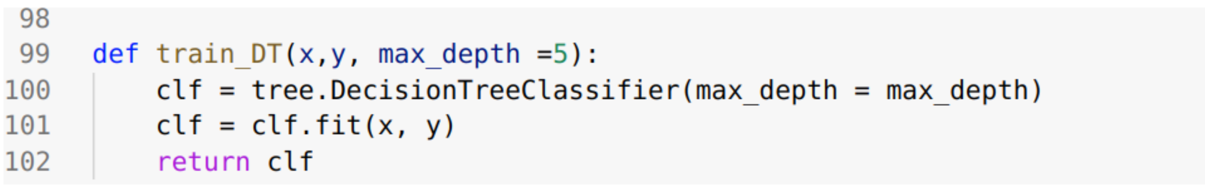
- Final data format. (H) (description, location, user guide):
    - First Some of the choices I made (so far):
        - The Satellite Images are only made up of the 13 Reflectance Bands Described Earlier (I Did not use any additional -Manually engineered and calculated- features such as NDVI or Cloud-Cover percentage)
        - As described above, these 12 Bands have different resolution: 4 bands have the high 10m resolutions, 6 bands are at 20m, and 3 have 60m resolution (per pixel). The 20m and 60m bands are upscaled to 10m resolution using bilinear interpolation.
        - The Sources we have so far (Gaziera, Managil, and Qadarif) are for the 2021 Season. However, Only for the Qdarif source, there is information about the 2020 and 2019 seasons. I did not use this information yet and the shared processed files are only for the 2021 season across all sources.
        - As our data is collected from different sources, each source has its own way of naming and assigning id to its field. However, the Field_Id Column in the final CSV is created by me and is unique across all sources (created by concatenating all fields from all sources and assigning ids as a series of integers.)
        - I chose to download the images from June to December, taking on the image around the mid of every month (7 images for each month) based on estimations of the agricultural season.
    - Interactive plot of the crops can be found on the Colab:



    - Each of the sources Dataframes have 83 columns described as below:
        - 1. Geometry: The (lat, long) point corresponding to that location (This is used for visualization)
        - 2. Field_Id: The id of the field from which the pixel is taken
        - 3. Crop_Type: The Crop in that pixel (This is used as the labels later)

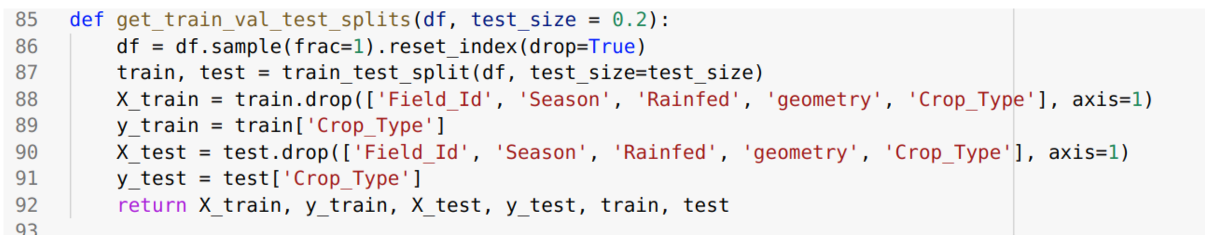
        - 4. Season: The year of the crop (this is all 2021as mentioned earlier)
        - 5. Rainfed: The type of watering for this pixel (0: irrigated, 1:rainfed)
        - 6-18. Band{1-13}_2021_12-15: The 13 bands for this pixel at the date 2021_12_15
        - 19-31. Band{1-13}_2021_11-15: The 13 bands for this pixel at the date 2021_11_15
        - …
        - …
        - 58-70. Band{1-13}_2021_12-15: The 13 bands for this pixel at the date 2021_07_15
        - 71-83. Band{1-13}_2021_12-15: The 13 bands for this pixel at the date 2021_06_15
    - The Columns from 6 to 83 are all used for training the model
    - The Crop_Type column is used as Labels.
    - the other columns are just for visualization and better understanding of the data.

6. Methods (H) :(description, parameters, features, bottlenecks, libraries/tech, data split)
    - Models: In this work, we leverage for our task.The decision tree classifier creates the classification model by building a decision tree.



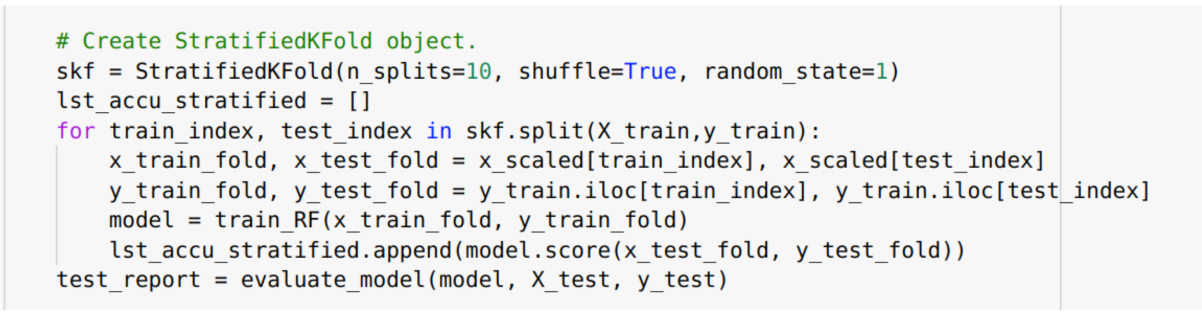
```python

def train_DT(x,y, max_depth =5):
    clf = tree.DecisionTreeClassifier(max_depth = max_depth)
    clf = clf.fit(x, y)
    return clf
```

Training: The input of our mode is Data frame (),we split our entire dataset into a training set %80 and a test set %20 using the train_test_split() class present in sklearn. Then we train our model on the training set and test our model on the test set.



```python
def get_train_val_test_splits(df, test_size = 0.2):
    df = df.sample(frac=1).reset_index(drop=True)
    train, test = train_test_split(df, test_size=test_size)
    X_train = train.drop(['Field_Id', 'Season', 'Rainfed', 'geometry', 'Crop_Type'], axis=1)
    y_train = train['Crop_Type']
    X_test = test.drop(['Field_Id', 'Season', 'Rainfed', 'geometry', 'Crop_Type'], axis=1)
    y_test = test['Crop_Type']
    return X_train, y_train, X_test, y_test, train, test
```

 We divide our training set to 10 Folds using Stratified K-Fold Cross-Validation. we train our model in each fold



```
# Create StratifiedKFold object.
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
lst_accu_stratified = []
for train_index, test_index in skf.split(X_train,y_train):
    x_train_fold, x_test_fold = x_scaled[train_index], x_scaled[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.iloc[test_index]
    model = train_RF(x_train_fold, y_train_fold)
    lst_accu_stratified.append(model.score(x_test_fold, y_test_fold))
test_report = evaluate_model(model, X_test, y_test)
```

7. Validation:
8. Testing:

9. Results and Discussion (H):
10. Conclusion:
11. References:

****

AMMAR, READ THIS PLEASE AND LET ME KNOW

****

In ML when we want to train our model we split our entire dataset into a training set and a test set in sklearn. this way has two down sides:
1- Changing the random_state parameter present in train_test_split(), leads to different accuracy for different random_state.
2- The train_test_split() splits the dataset by random sampling.
we can face many issues and overfitting is one of them

<u>ML Techniques to Prevent Overfitting</u> :

1. More Data for Better Signal Detection
2. early stopping
3. cross validation

I do recommend using stratified K-fold cross validation , the only difference between it and K-fold cross validation is that it does stratified sampling instead of random sampling. to ensure that each fold of dataset has the same proportion of observations with a given label.

- **I WILL SHARE SOME CODE FOR STRATIFIED K- FOLD CROSS VALIDATION WITH YOU.**

Tasks1:

Model & Parameter used:

- **Decision Tree. (Depth=5)**
- **No under sampling.**
- **Metric: Accuracy**
- **n_splits=10**
- **Test Size = 20%, Train Size = 80%**

- **Train the model on the whole data using k-fold cross validation**
- **Expected Results :**
  - **List of ‘n_splits ’Accurcies for the folds.**
  - **Comprae these accuracise against Test Set**
- **Expriment Report:**

Split number	**Trian Acc**	**Test Acc**

1		
2		
3		
4		
5		
6		
7		
8		
9		
10		