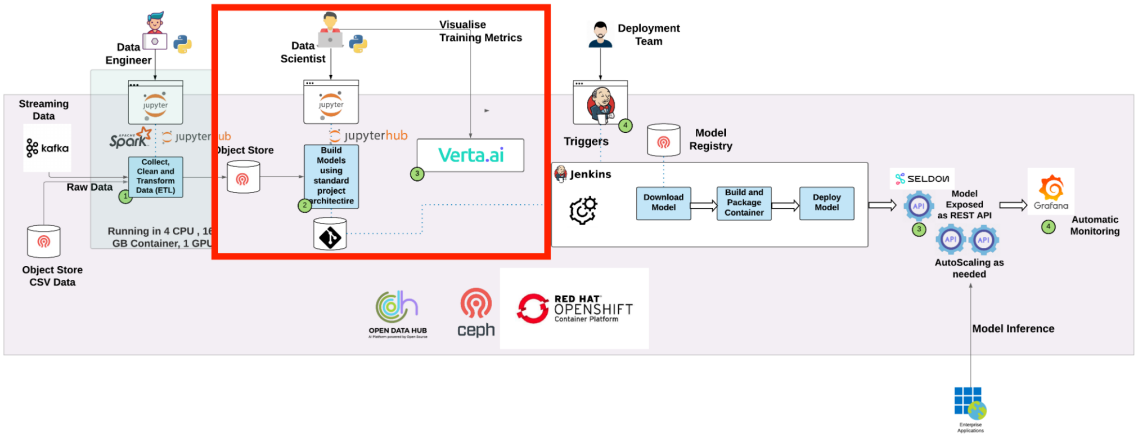
Lab 2 - Data Science

Introduction

Next we feature three Jupyter notebooks the data scientist uses, pulling the prepared CSV data that the data engineer pushed to S3 object storage in the previous lab:

1. They first visualise the data - to understand patterns in the data and whether there are any errors they need to fix before experimenting and training their models.
2. They then experiment with different algorithms, parameters and hyperparameters. They push each experiment to the model repository, Verta. This repository contains all of the data and the actual model binaries should they wish to
    a. Compare different experiments
    b. Return to and retrieve any experiment they ran
    c. Share their experiments with others. In this way, we're allowing silos between different actors in the workflow
3. They then choose one of their experiments, which they wish to proceed with and push to production - in the next part of the workflow, the ML OPs phase

This diagram illustrates the workflow we're implementing - the Data Science part of the overall AI/ML workflow:



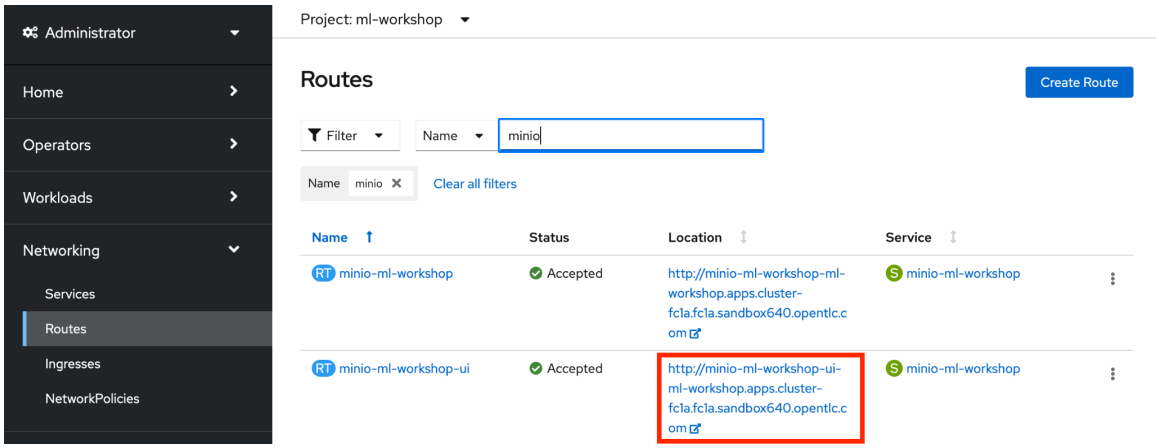
Instructions to access your prepared data file from the previous lab

You need to access the prepared CSV data file you created and pushed to S3 object storage, in the previous lab under the Data Engineer persona.

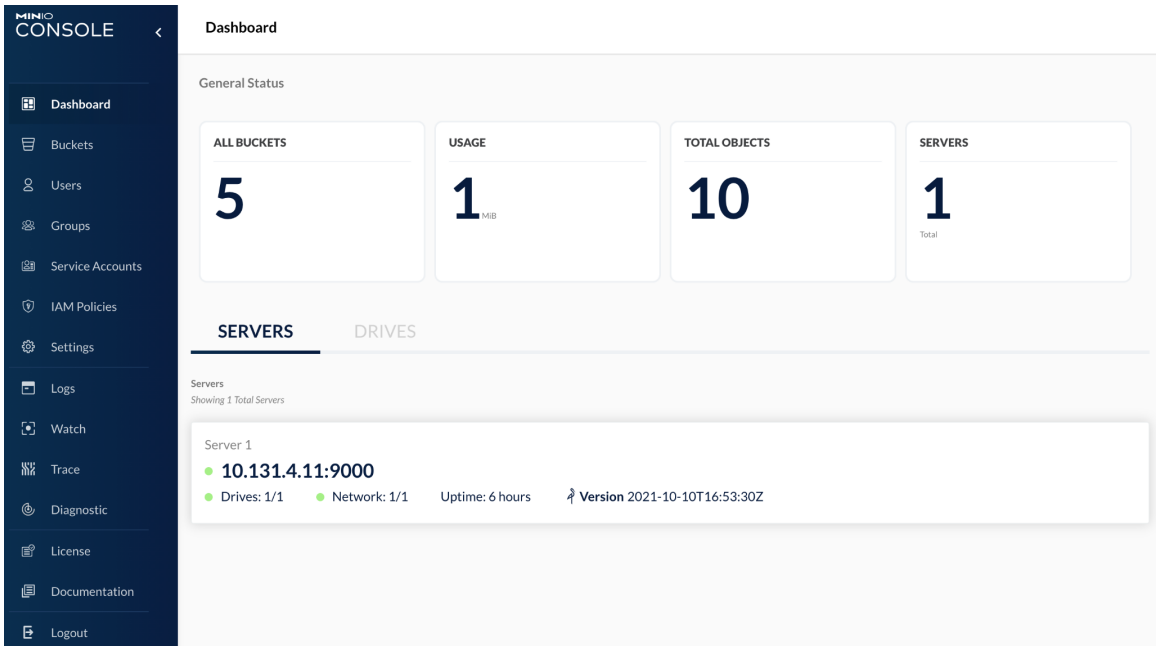
Login to OpenShift using the credentials your administrator gave you. Ensure your workshop project ml-workshop is selected.

The first thing you need to do is retrieve the path and file pertaining to your username - which you as a data engineer created previously.
Choose the **Administration perspective**

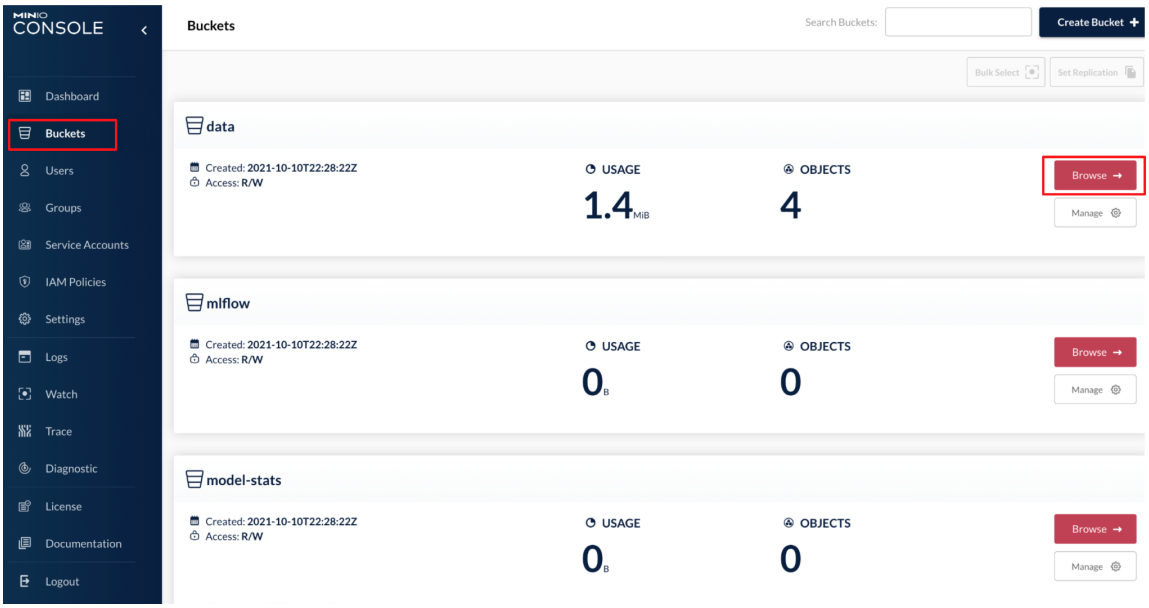
1. Navigate to **Networking > Routes**.
2. Filter on *minio* - and open the *minio-ml-workshop-ui* route as shown.



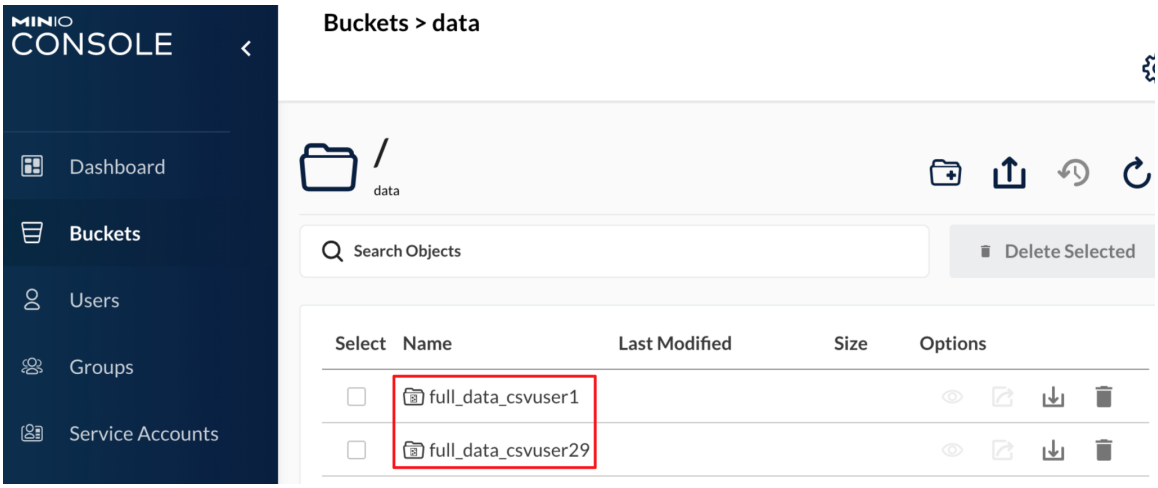
3. Enter the username and password minio / minio123



4. Click **Buckets** in the left panel to display your storage buckets.
5. Locate the **data** bucket, and click the **Browse**.



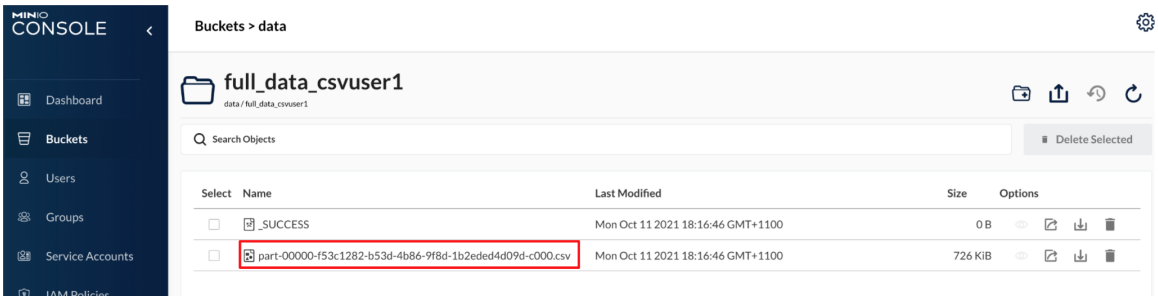
Minio will display the files for everyone doing this lab.



6. Locate and click the file **fuill_data_csvuser** that has your username. E.g. **full_data_csvuser29**.

Note: There will be many folders there - be sure to select the one containing **your username.**

7. Click the long filename:



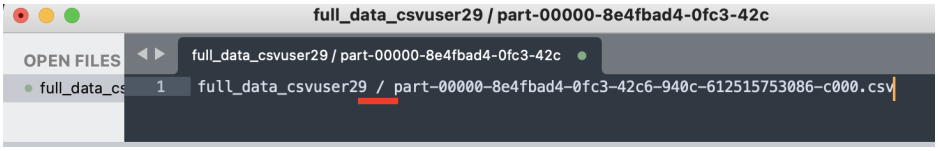
Minio displays the details of the file you created in the Data Engineering lab. Here you can retrieve your CSV file - which we'll refer to throughout this section as **YOUR_CSV_FILE.** To do this select your folder and filename as shown:

8. Highlight the filename path as shown below and copy the text to the clipboard. E.g.
`full_data_csvuser1 / part-00000-f53c1282-b53d-4b86-9f8d-1b2eded4d09d-c000.csv`

9. Open any editor you choose and paste this into a text editor so you can use it later. E.g. Sublime Text or Microsoft Word.

Note: Minio includes two spaces that you need to remove so your code will run.

10. Delete the two embedded spaces on either side of the forward slash:



You file path should now look similar to this:

Keep this file open for the 3 data science focused notebook exercises. As mentioned we'll refer to this long filename string as **YOUR_CSV_FILE** - which you'll paste into your notebooks below**.**

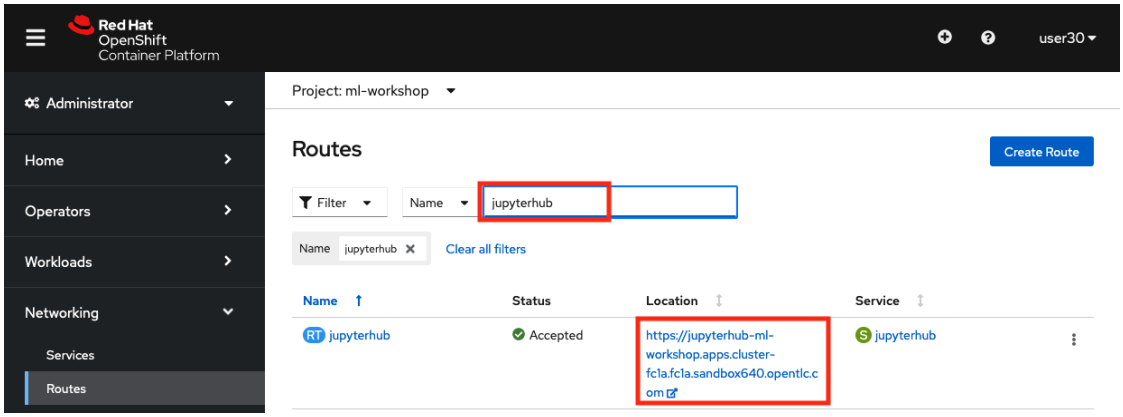
Part 1: Visualise Data

Now to our Data Science focused Jupyter notebooks. As we did with Minio, we will find the url for Jupyterhub within the routes of the OPenShift console.

1. Open the browser tab with the OpenShift console.
2. Open the **Administrator perspective**.
3. Click **Networking > Routes**.
4. Type **Jupyterhub** in the **Filter** text box.

OpenShift reduces the list of routes as you type the filter.

5. Click the Jupyterhub link in the **Location** column of the **Routes** display.



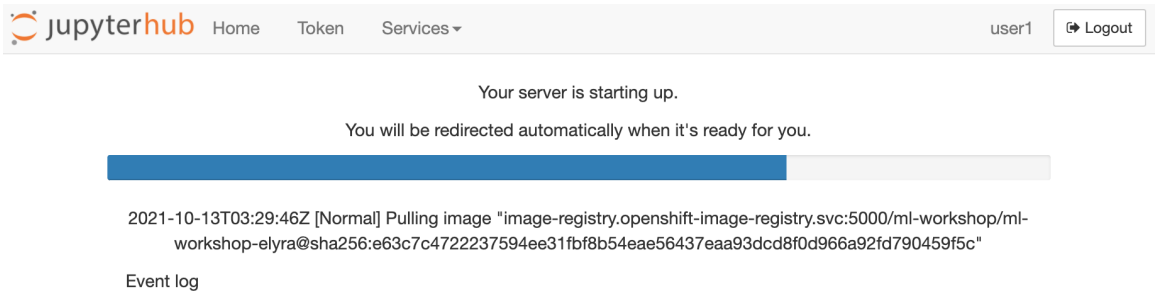
Because you shutdown your Jupyter server at the end of the last workshop, you'll again be presented with the Jupyter screen where you choose the base image to work with. As we're now assuming the role of a data scientist, you'll choose:

- A different base image, the *MLWorkShop Notebook Image*
- A *Large* container - allocating the maximum amount of CPU and memory available.

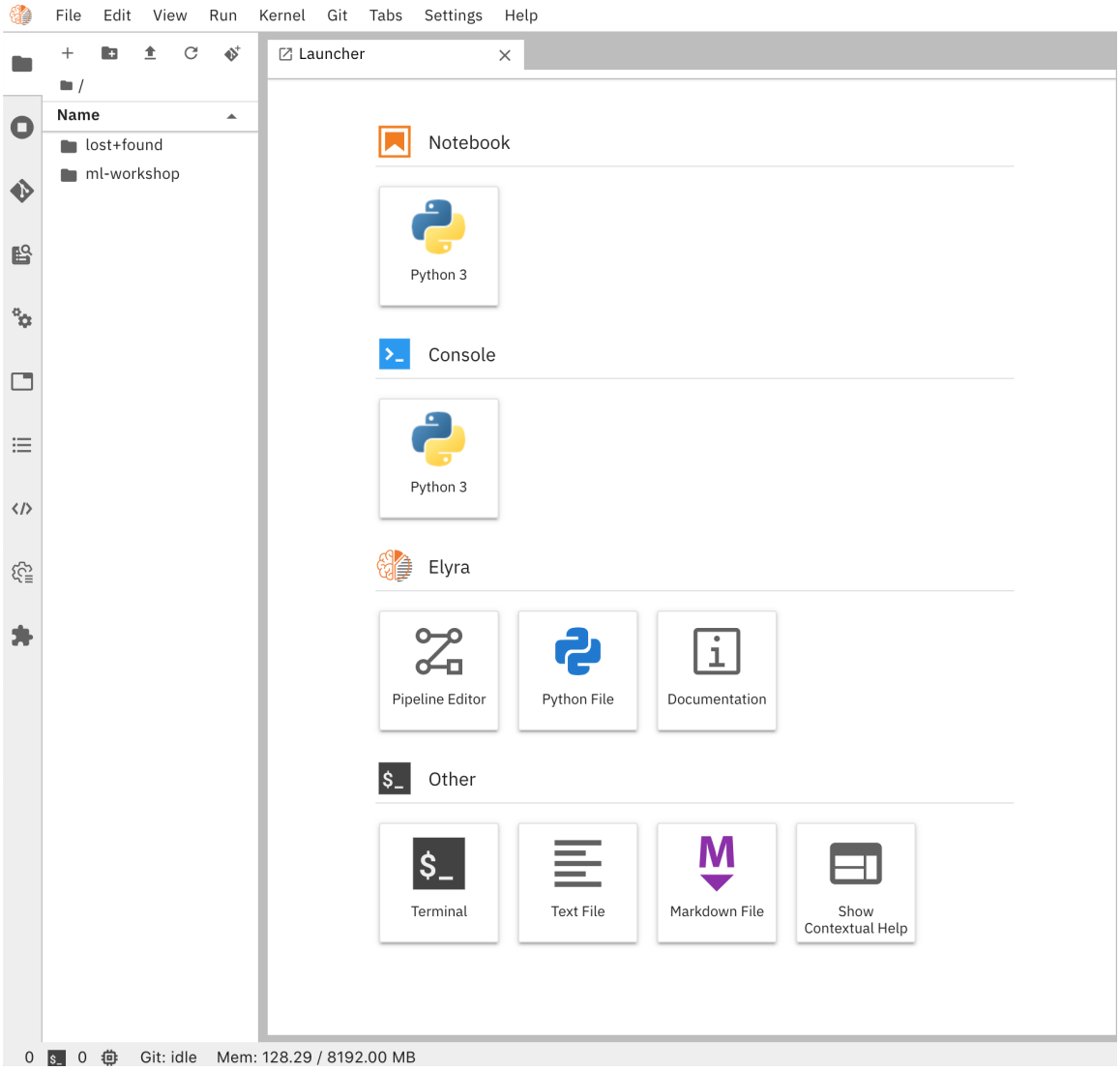
Warning: Please select the correct notebook image, otherwise the lab will not work.

6. Click the **MLWorkshop Notebook Image** radio button
7. Click the **Container size** dropdown list box and select **Large**.
8. Click **Start Server**.

Jupyterhub starts the notebook server for the Data Scientist.



After a few minutes the notebook will have started and the Jupyter notebook will be displayed.



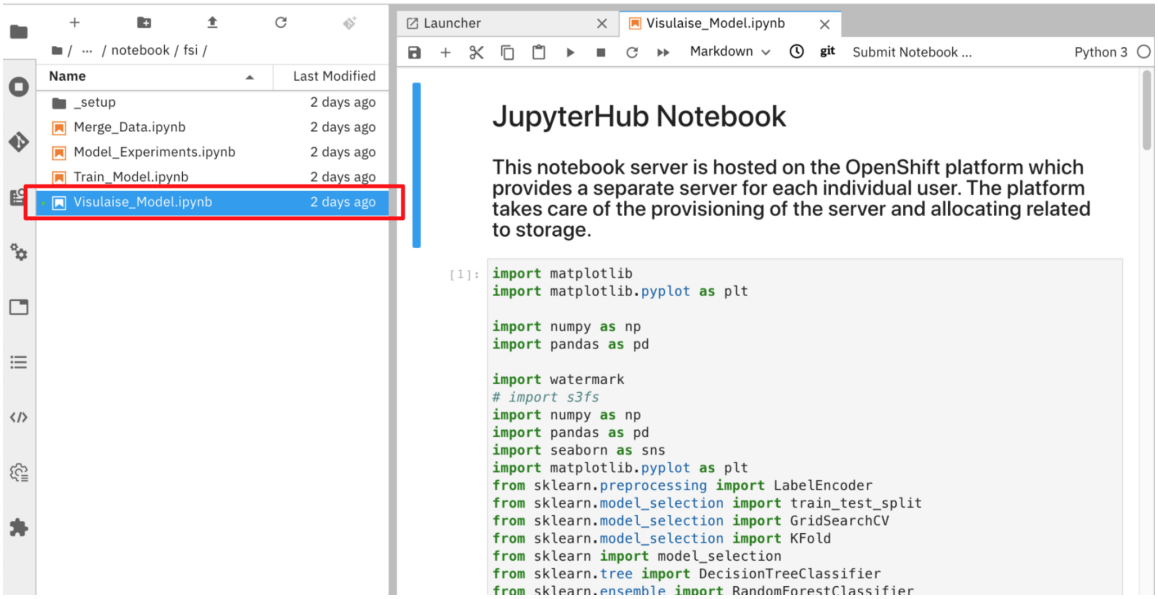
Observe:

- The different look and feel of the user interface. This is a more up to date version known as *Elyra*.
- The same *ml-workshop* folder we previously pulled down from our GitHub repository https://github.com/masoodfaisal/ml-workshop

9. Using the File Explorer in the Jupyter notebook, navigate to the project files you will use for this lab:
   For Financial Services labs:
   - **ml-workshop > notebook > fsi**

   For Telecommunications labs:
   - **ml-workshop > notebook > telco**

10. Double click the file **Visulaise_Model.ipynb** as shown
    (The name is a little misleading - it's really there to visualise the data not the model.)



Before we get going we need to update the filename used in the lab to match your file from lab 1.

11. Scroll down to cell number **[4]**.
12. Paste the filename of **YOUR_CSV_FILE** that you saved in lab 1 between the quotes as illustrated in the following diagram.
    Make sure you have removed the spaces either side of the /.

13. Click **File > Save Notebook** to save your work.

In this next section, on the second line, insert the value you retrieved from Minio object storage earlier - representing the fully qualified name of your csv file in Minio. This is the file pushed by the data engineer in the format: full_data_csv{USERNAME}/{FILENAME}.csv.

In my case this value is: full_data_csvuser29/part-00000-59149e08-583c-46a5-bfa0-0b3abecbf1a3-c000.csv (yours will be different)

We refer to this fully qualified name in the Github instructions as CSV-FILE

```
[4]: minioClient = get_s3_server()
data_file = minioClient.fget_object("data", "full_data_csvuser1/part-00000-f53c1282-b53d-4b86-9f8d-1b2eded4d09d-c000.csv", "/tmp/data.csv")
data_file_version = data_file.version_id
data = pd.read_csv('/tmp/data.csv')
data.head(5)
```

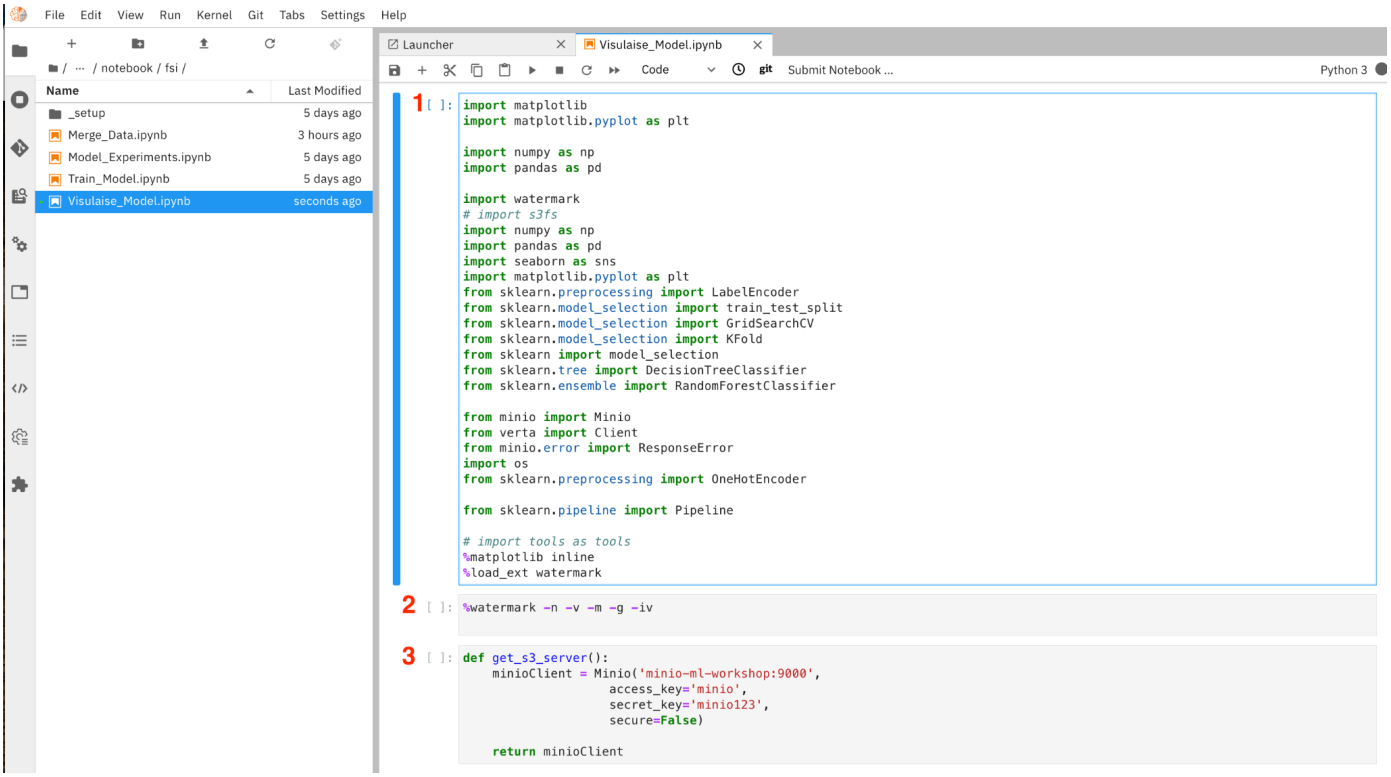
[4]: customerID gender SeniorCitizen Partner Dependents tenure Premium RelationshipManager PrimaryChannel HasCreditCard ... IncomeProtection WealthManagement HomeEq

14. Scroll up to the top of the notebook
15. Click in cell **[1]**.

You will now step through the notebook one cell at a time.

16. Type **[Shift] + [Return]** to step through each cell in the notebook.

Now, as previously, select the first cell and walk through each cell executing you go by clicking SHIFT + RETURN. Make sure you **pause** and make the changes **as indicated in red**, before executing certain cells.



1. Import our desired Python libraries. (notice we don't need to do any ***pip install***s - our administrator has bundled all of our required libraries into this base container image - which we selected earlier the *MLWorkShop Notebook Image*)
2. *watermark* outputs the versions of various components, libraries, operating system attributes etc.
3. Here we connect to our S3 object store, Minio, using the URL and credentials shown

4

```
minioClient = get_s3_server()
data_file = minioClient.fget_object("data", "full_data_csvuser29/part-00000-8a222f86-81cd-49e5-bc60-e28e3ac60d65-c000.csv", "/tmp/data.csv")
data_file_version = data_file.version_id
data = pd.read_csv('/tmp/data.csv')
data.head(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Premium	RelationshipManager	PrimaryChannel	HasCreditCard	...	IncomeProtection	WealthManagement	HomeEqui
0	148	Male	0	No	No	1	Yes	No	Mobile	No	...	No	No	
1	463	Male	0	Yes	Yes	4	Yes	Yes	Branch	No	...	Yes	No	
2	471	Female	1	No	No	17	Yes	No	No	Not Available	...	Not Available	Not Available	Not A
3	496	Male	0	No	No	22	No	Not available	Mobile	No	...	Yes	No	
4	833	Female	0	Yes	Yes	70	Yes	No	Mobile	Yes	...	Yes	Yes	

5 rows × 21 columns

Use pandas.DataFrame functions

- *shape* to return the dimensionality
- *info* to print a concise summary of the DataFrame
- *describe* to generate descriptive statistics of the DataFrame's columns
- *isnull().sum()* to sum the empty values
- finally determine Churn and Total Changes

5

```
data.shape
```

```
(7043, 21)
```

6

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   customerID           7043 non-null   int64
 1   gender               7043 non-null   object
 2   SeniorCitizen        7043 non-null   int64
 3   Partner              7043 non-null   object
 4   Dependents           7043 non-null   object
 5   tenure               7043 non-null   int64
 6   Premium              6978 non-null   object
 7   RelationshipManager  6978 non-null   object
 8   PrimaryChannel       6978 non-null   object
 9   HasCreditCard        6978 non-null   object
 10  DebitCard            6978 non-null   object
 11  IncomeProtection     6978 non-null   object
 12  WealthManagement     6978 non-null   object
 13  HomeEquityLoans      6978 non-null   object
 14  MoneyMarketAccount   6978 non-null   object
 15  CreditRating         6978 non-null   object
 16  PaperlessBilling     6978 non-null   object
 17  AccountType          6978 non-null   object
 18  MonthlyCharges       6978 non-null   float64
 19  TotalCharges         6967 non-null   float64
 20  Churn                6978 non-null   object
dtypes: float64(2), int64(3), object(16)
memory usage: 1.1+ MB
```

7

```
data.describe()
```

	customerID	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	6978.000000	6967.000000
mean	3522.000000	0.162147	32.371149	64.706614	2280.638015

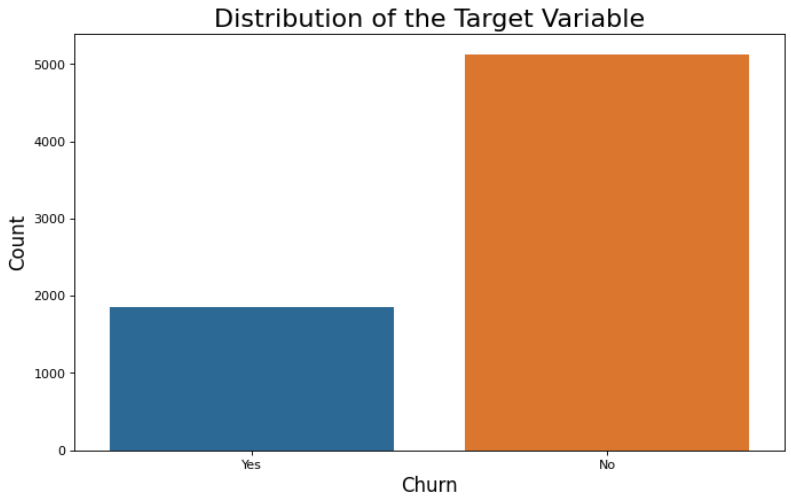
4. In this cell we also output the first 5 lines of the file - so the data scientist can get a quick view of the data.
5. We output the dimensions of the data in rows and columns (features)
6. Here we output various data around the columns (features) including their types, names etc
7. Using *describe(),* we output various statistical data associated with the entire dataset, max, mean etc. values for numeric columns.

8
```
[8]: data.isnull().sum()
```

```
[8]: customerID              0
     gender                  0
     SeniorCitizen           0
     Partner                 0
     Dependents              0
     tenure                  0
     Premium                65
     RelationshipManager    65
     PrimaryChannel         65
     HasCreditCard          65
     DebitCard              65
     IncomeProtection       65
     WealthManagement       65
     HomeEquityLoans        65
     MoneyMarketAccount     65
     CreditRating           65
     PaperlessBilling       65
     AccountType            65
     MonthlyCharges         65
     TotalCharges           76
     Churn                  65
     dtype: int64
```

9
```
[9]: fig = plt.figure(figsize=(10,6), dpi=80)
     ax = sns.countplot(x="Churn", data=data)
     ax.set_title('Distribution of the Target Variable', fontsize=20)
     ax.set_xlabel('Churn', fontsize = 15)
     ax.set_ylabel('Count', fontsize = 15)
```

```
[9]: Text(0, 0.5, 'Count')
```



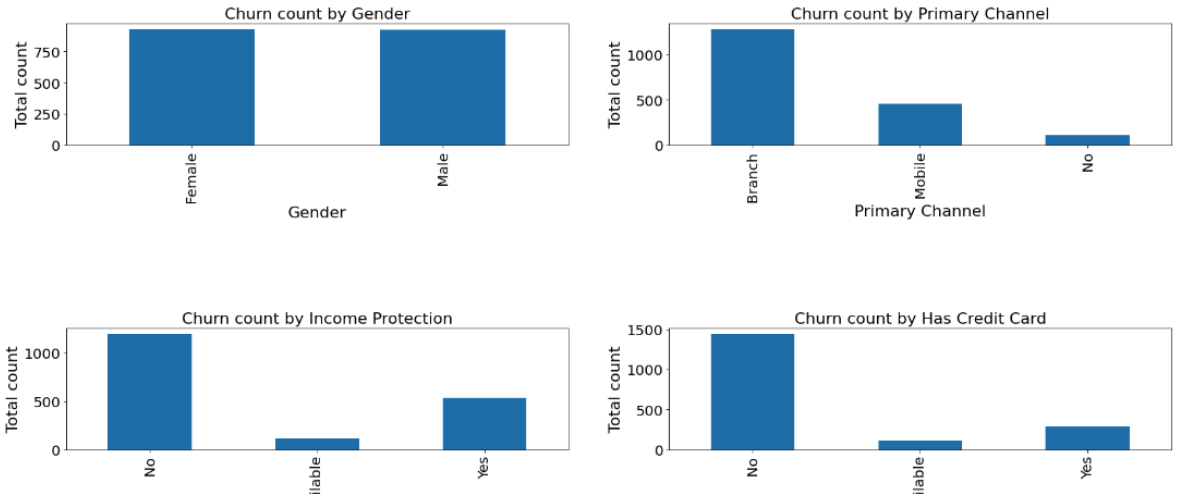
10
```
[10]: # Convert binary variable into numeric so plotting is easier. We need to later take mean
      data['Churn'] = data['Churn'].map({'Yes': 1, 'No': 0})
```

8. We output the sum of rows with null values with nulls - to assess data for errors, e.g null for *charges* indicates an error.
9. Here we output the total count of the ***labeled*** column, Churn. We need a decent spread, and we have it - with just over 2 to 1.
10. Here we make a simple conversion from Yes and No to 1 and 0, to facilitate plotting.

11

```
[11]: fig, ((ax1,ax2),(ax3,ax4), (ax5,ax6)) = plt.subplots(ncols=2, nrows=3, figsize=(25,17), dpi = 80)
plt.subplots_adjust(left=None, bottom=None, right=None, top=None ,wspace=None, hspace=1.5)
plt.rc('xtick', labelsize = 12)    # fontsize of the tick labels
plt.rc('ytick', labelsize = 12)

data.groupby('gender').Churn.sum().plot(kind='bar', ax = ax1)
ax1.set_ylabel('Total count',fontsize = 20)
ax1.set_xlabel('Gender',fontsize = 20)
ax1.tick_params(labelsize = 18)
ax1.set_title('Churn count by Gender',fontsize = 20)

data.groupby('PrimaryChannel').Churn.sum().plot(kind='bar', ax=ax2)
ax2.set_ylabel('Total count',fontsize = 20)
ax2.set_xlabel('Primary Channel',fontsize = 20)
ax2.tick_params(labelsize = 18)
ax2.set_title('Churn count by Primary Channel',fontsize = 20)

data.groupby('IncomeProtection').Churn.sum().plot(kind='bar', ax=ax3)
ax3.set_ylabel('Total count',fontsize = 20)
ax3.set_xlabel('Income Protection',fontsize = 20)
ax3.tick_params(labelsize = 18)
ax3.set_title('Churn count by Income Protection',fontsize = 20)

data.groupby('HasCreditCard').Churn.sum().plot(kind='bar', ax=ax4)
ax4.set_ylabel('Total count',fontsize = 20)
ax4.set_xlabel('Has Credit Card',fontsize = 20)
ax4.tick_params(labelsize = 18)
ax4.set_title('Churn count by Has Credit Card',fontsize = 20)

data.groupby('WealthManagement').Churn.sum().plot(kind='bar',ax=ax5)
ax5.set_ylabel('Total count',fontsize = 20)
ax5.set_xlabel('Wealth Management',fontsize = 20)
ax5.tick_params(labelsize = 18)
ax5.set_title('Churn count by Wealth Management',fontsize = 20)

data.groupby('CreditRating').Churn.sum().plot(kind='bar',ax=ax6)
ax6.set_ylabel('Total count',fontsize = 20)
ax6.set_xlabel('Credit Rating Type',fontsize = 20)
ax6.tick_params(labelsize = 18)
ax6.set_title('Churn count by Credit Rating',fontsize = 20)
```

```
[11]: Text(0.5, 1.0, 'Churn count by Credit Rating')
```



11. This cell visually outputs churn count by various features in the data set

12
```
data.replace(" ", np.nan, inplace=True)
```

13
```
data.isna().sum()
```

```
customerID              0
gender                  0
SeniorCitizen           0
Partner                 0
Dependents              0
tenure                  0
Premium                65
RelationshipManager    65
PrimaryChannel         65
HasCreditCard          65
DebitCard              65
IncomeProtection       65
WealthManagement       65
HomeEquityLoans        65
MoneyMarketAccount     65
CreditRating           65
PaperlessBilling       65
AccountType            65
MonthlyCharges         65
TotalCharges           76
Churn                  65
dtype: int64
```

14
```
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'])
```

15
```
mean = data['TotalCharges'].mean()
data.fillna(mean, inplace=True)
# Now we know that total charges has nan values
data.isna().sum()
```

```
customerID             0
gender                 0
SeniorCitizen          0
Partner                0
Dependents             0
tenure                 0
Premium                0
RelationshipManager    0
PrimaryChannel         0
HasCreditCard          0
DebitCard              0
IncomeProtection       0
WealthManagement       0
HomeEquityLoans        0
MoneyMarketAccount     0
CreditRating           0
PaperlessBilling       0
AccountType            0
MonthlyCharges         0
TotalCharges           0
Churn                  0
dtype: int64
```

16
```
plt.figure(figsize=(10,8), dpi=80)
# Sns.set(rc={'figure.figsize':(25,15)})
ax = sns.catplot(x="CreditRating", y="TotalCharges", hue="Churn", kind="box", data=data, height = 6,aspect = 1.5,palette = 'RdBu')
plt.title('Comparison of Total Charges for each CreditRating',fontsize = 20)
plt.xlabel('CreditRating',fontsize = 15)
plt.ylabel('Total Charges',fontsize = 15)
```

```
Text(17.690962577160498, 0.5, 'Total Charges')
<Figure size 800x640 with 0 Axes>
```

12. Replace spaces with numpy NAN values
13. Output sum of NAN and None values
14. Convert to numeric
15. Fill NANs with the mean
16. Here we output a box plot - a useful visualization of 2 dimensions by the labeled column *Churn*

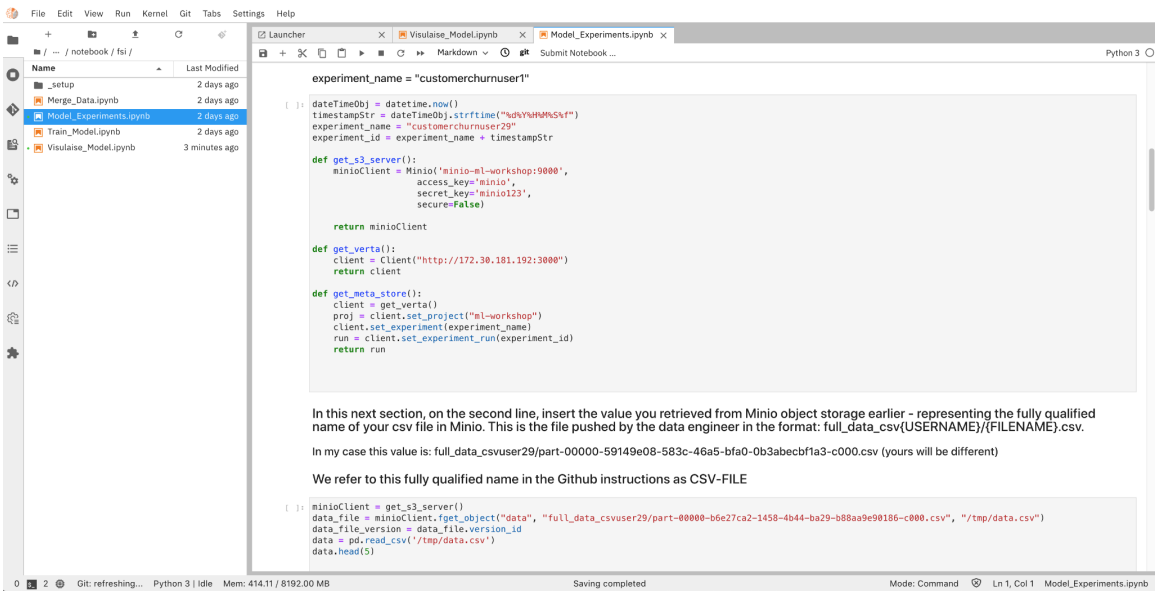
Part 2: Experiment with Models

At this point, as a data scientist, we have a good understanding of the data. Now it's time to start experimenting with different models, parameters and hyper parameters.

As we experiment, we want our notebook to create an experiment id for every experiment (which is guaranteed to be unique within our team, as it uses user id and timestamp as a basis).

This experiment id is then used as an identifier when we push our experiment metadata and binaries to our model repository, Verta. In this way, we retrieve and repeat any experiment we have done, as well as share this experiment with other team members, breaking down silos between teams and individuals in AI//ML workflows.

1. Using the File Explorer, open the **Model_Experiments.ipynb** notebook. Jupyterhub opens the code windows.



Next, just as you did in the previous lab you will step through the code one cell at a time. **Note there are 3 changes you'll need to make to your cells - which we'll highlight below.**

2. Scroll up to the top of the notebook
3. Click in cell **[1]**.

You will now step through the notebook one cell at a time.

4. Type **[Shift] + [Return]** to step through each cell in the notebook.

Make sure you **pause** and make the changes **as indicated in red**, before executing certain cells.

[1]:
```
import os
# os.environ["MODIN_ENGINE"] = "ray"
```

[2]:
```
import matplotlib
import matplotlib.pyplot as plt

import numpy as np
# import pandas as pd
# import modin.pandas as pd

import watermark
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from datetime import datetime
import verta.integrations.sklearn
from minio import Minio
from verta import Client
from minio.error import ResponseError
import os
from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline


# import tools as tools
%matplotlib inline
%load_ext watermark
```

[3]:
```
%watermark -n -v -m -g -iv
```

1. Do imports - more applicable when we want to use *Ray*.
2. More imports of libraries we need
3. See *watermark* description of this cell above in - ***first Data Science workshop - Visualisation***

4

```
dateTimeObj = datetime.now()
timestampStr = dateTimeObj.strftime("%d%Y%H%M%S%f")
experiment_name = "customerchurnuser29"
experiment_id = experiment_name + timestampStr

def get_s3_server():
    minioClient = Minio('minio-ml-workshop:9000',
                        access_key='minio',
                        secret_key='minio123',
                        secure=False)

    return minioClient

def get_verta():
    client = Client("http://172.30.46.176:3000")
    return client

def get_meta_store():
    client = get_verta()
    proj = client.set_project("ml-workshop")
    client.set_experiment(experiment_name)
    run = client.set_experiment_run(experiment_id)
    return run
```

In this next section, on the second line, insert the value you retrieved from Minio object st
name of your csv file in Minio. This is the file pushed by the data engineer in the format: fu

In my case this value is: full_data_csvuser29/part-00000-59149e08-583c-46a5-bfa0-0b3abecbf1a3-c

We refer to this fully qualified name in the Github instructions as CSV-FILE

5

```
minioClient = get_s3_server()
data_file = minioClient.fget_object("data", "full_data_csvuser29/part-00000-8a222f86-81cd-49e5-bc60-e
data_file_version = data_file.version_id
data = pd.read_csv('/tmp/data.csv')
data.head(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Premium	RelationshipManager	PrimaryChannel	HasC
0	148	Male	0	No	No	1	Yes	No	Mobile	
1	463	Male	0	Yes	Yes	4	Yes	Yes	Branch	
2	471	Female	1	No	No	17	Yes	No	No	No
3	496	Male	0	No	No	22	No	Not available	Mobile	
4	833	Female	0	Yes	Yes	70	Yes	No	Mobile	

5 rows × 21 columns

Use pandas.DataFrame functions

- *shape* to return the dimensionality
- *info* to print a concise summary of the DataFrame
- *describe* to generate descriptive statistics of the DataFrame's columns
- *isnull().sum()* to sum the empty values
- finally determine Churn and Total Changes

6

```
data.shape
```

```
(7043, 21)
```

7

```
data.info()
```

4. **You need to modify this cell before you execute it.**
   Replace the third line replacing user29 with your userid. Save the file
   Next we connect to Minio to pull our raw data from.
   Also connect to Verta to push our experiment data to. Set project within Verta to be ml-workshop.
5. **You need to modify this cell before you execute it.**
   Here we retrieve the CSV we prepared in the Data Engineer lab earlier. To do that replace the string that's underlined in the screenshot with the string representing your file, **YOUR_CSV_FILE** from earlier in this lab. Save the file and continue executing.
   In this cell we also output the first 5 lines of the file - so the data scientist can get a quick view of the data.
6. see *data.shape* cell description above in - ***first Data Science workshop - Visualisation***
7. see *data.info* cell description above in - ***first Data Science workshop - Visualisation***

Run all the way down to cell 15, *Feature Engineering Pipeline*, as all cells until then are discussed above in - ***first Data Science workshop - Visualisation***

15

```
import category_encoders as ce
import joblib

names = ['gender', 'Partner', 'Dependents', 'Premium', 'HomeEquityLoans', 'MoneyMarketAccount', 'PaperlessBilling', 'Churn']
# for column in names:
#     labelencoder(column)

enc = ce.ordinal.OrdinalEncoder(cols=names)
enc.fit(data)
joblib.dump(enc, 'enc.pkl')
labelled_set = enc.transform(data)
labelled_set.tail(5)
```

```
/opt/app-root/lib/python3.6/site-packages/category_encoders/utils.py:21: FutureWarning: is_categorical is deprecated and will be removed in
on.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Premium	RelationshipManager	PrimaryChannel	HasCreditCard	...	IncomeProtection	WealthMan
7038	6490	1	0	1	1	1	1	No	No	Not Available	...	Not Available	Not
7039	6634	2	0	1	1	1	1	Yes	Branch	No	...	No	
7040	6638	1	0	2	1	69	1	No	Mobile	Yes	...	No	
7041	6721	1	0	2	2	70	1	Yes	Mobile	No	...	Yes	
7042	6819	2	0	1	1	3	1	No	Mobile	Yes	...	No	

5 rows × 21 columns

16

```
names = ['RelationshipManager', 'PrimaryChannel', 'CreditRating', 'AccountType', 'HasCreditCard', 'DebitCard',
         'IncomeProtection', 'WealthManagement']

ohe = ce.OneHotEncoder(cols=names)
ohe.fit(labelled_set)
joblib.dump(ohe, 'ohe.pkl')
final_set = ohe.transform(labelled_set)
final_set.tail(5)
```

```
/opt/app-root/lib/python3.6/site-packages/category_encoders/utils.py:21: FutureWarning: is_categorical is deprecated and will be removed in
on.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Premium	RelationshipManager_1	RelationshipManager_2	RelationshipManager_3	...	CreditRating
7038	6490	1	0	1	1	1	1	1	0	0	...	
7039	6634	2	0	1	1	1	1	0	1	0	...	
7040	6638	1	0	2	1	69	1	1	0	0	...	
7041	6721	1	0	2	2	70	1	0	1	0	...	
7042	6819	2	0	1	1	3	1	1	0	0	...	

5 rows × 46 columns

17

Now we use scikit-learn's 'train_test_split' function to randomly split our data into training and testing sets. Then remove the and *customerID* fields from our training and testing datasets and output the shaope of our data.

```
labels = final_set['Churn']
X_train, X_test, y_train, y_test = train_test_split(final_set, labels, test_size=0.2)
X_train.pop('Churn')
X_train.pop('customerID')
X_test.pop('Churn')
X_test.pop('customerID')
print ('Training Data Shape',X_train.shape, y_train.shape)
print ('Testing Data Shape',X_test.shape, y_test.shape)
```

```
Training Data Shape (5634, 44) (5634,)
Testing Data Shape (1409, 44) (1409,)
```

18

```
# Data For cross validation and GridSearch
Y = final_set['Churn']
X = final_set.drop(['Churn', 'customerID'], axis=1)
print ('Training Data Shape', X.shape)
print ('Testing Data Shape', Y.shape)
```

15. Here we use an Ordinal Encoder to convert simple binary values to a numeric representation. Output the data after applying the Ordinal Encoder.
16. Here we use a One Hot Encoder to convert multi valued features to a numeric representation. Output the data after applying the One Hot Encoder.
17. Here we split our data set into a training and a testing set, and discard unwanted columns customer id and our labeled column Churn.
18. Further data set refinement.

Create DecisionTreeClassifier object, extract hyper parameters, and then GridSearch will best_mod

19
```
# Create decision tree object
DT = DecisionTreeClassifier()
# List of parameters
# entropy
criterion = ['gini']
max_depth = [5,10,15]
min_samples_split = [2,4,6]
min_samples_leaf = [4,5,6,8]
# Save all the lists in the variable
hyperparameters = dict(max_depth=max_depth, criterion=criterion,min_samples_leaf = min_samples_leaf ,min_samples
```

20
```
model = GridSearchCV(DT, hyperparameters, cv=5, verbose=0)
best_model = model.fit(X,Y)
```

21
```
# Mean cross validated score
print('Mean Cross-Validated Score: ',best_model.best_score_)
print('Best Parameters',best_model.best_params_)
# You can also print the best penalty and C value individually from best_model.best_estimator_.get_params()
print('Best criteria:', best_model.best_estimator_.get_params()['criterion'])
print('Best depth:', best_model.best_estimator_.get_params()['max_depth'])
```

```
Mean Cross-Validated Score:  0.7936973756371378
Best Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 6, 'min_samples_split': 2}
Best criteria: gini
Best depth: 5
```

Use K-Folds cross-validator to split data in train/test sets. Create a dictionary of hyperparameter ca
DecisionTreeClassifier, assess results, print and store hyper parameters and accuracy and tag using

22
```
kfold = KFold(n_splits = 3)
hyperparameters = dict(max_depth=5, criterion='gini',min_samples_leaf = 3 ,min_samples_split = 10)
model = DecisionTreeClassifier(max_depth=5, criterion='gini',min_samples_leaf = 3 ,min_samples_split = 10)
model = model.fit(X_train, y_train)
joblib.dump(model, 'dct.pkl')
results = model_selection.cross_val_score(model,X,Y,cv = kfold)
print(results)
print('Accuracy',results.mean()*100)
store = get_meta_store()
store.log_hyperparameters(hyperparameters)
store.log_model(model)
store.log_metric('Accuracy',results.mean()*100)
store.log_tag("DecisionTreeClassifier")
# get_meta_store().log_dataset_version("raw_data", dataset_version)
```

```
[0.78321976 0.80579216 0.79037069]
Accuracy 79.31275370082314
connection successfully established
created new Project: ml-workshop in personal workspace
created new Experiment: customerchurnuser29
created new ExperimentRun: customerchurnuser29072021012111535042
upload complete (custom_modules)
upload complete (model.pkl)
upload complete (model_api.json)
```

Like before, in this next section, on the third line, change experiment_name by appending your use
your username is user1:

```
experiment_name = "customerchurnuser1"
```

19. Create a DecisionTreeClassifier with these hyper parameters
20. User GridSearch to output the best model / hyper parameters from the combinations supplied to its *fit* method.
21. Print out those best model parameters
22. Use K-Folds cross-validator to split data into train/test sets. Create a dictionary of hyperparameter candidates, train model using a DecisionTreeClassifier. Print and store hyper parameters and accuracy in Verta and tag using 'DecisionTreeClassifier''. The ***store*** method lushes this metadata to Verta, which you'll see below.

Create RandomForestClassifier object, extract hyper parameters, and then the best_model

23
```
dateTimeObj = datetime.now()
timestampStr = dateTimeObj.strftime("%d%Y%H%M%S%f")
experiment_name = "customerchurnuser29"
experiment_id = experiment_name + timestampStr


# Create random forest object
RF = RandomForestClassifier()
n_estimators = [18,22]
criterion = ['gini', 'entropy']
# Create a list of all of the parameters
max_depth = [30,40,50]
min_samples_split = [6,8]
min_samples_leaf = [8,10,12]
# Merge the list into the variable
hyperparameters = dict(n_estimators = n_estimators,max_depth=max_depth, criterion=criterion,min_samples_leaf = min_samples_l
# Fit your model using gridsearch
model = GridSearchCV(RF, hyperparameters, cv=5, verbose=0)
best_model = model.fit(X, Y)
```

Extract best scores, params, criteria and depth from our model.

24
```
# Mean cross validated score
print('Mean Cross-Validated Score: ',best_model.best_score_)
print('Best Parameters',best_model.best_params_)
# You can also print the best penalty and C value individually from best_model.best_estimator_.get_params()
print('Best criteria:', best_model.best_estimator_.get_params()['criterion'])
print('Best depth:', best_model.best_estimator_.get_params()['max_depth'])
print('Best estimator:', best_model.best_estimator_.get_params()['n_estimators'])
```

```
Mean Cross-Validated Score:  0.803778570391638
Best Parameters {'criterion': 'gini', 'max_depth': 50, 'min_samples_leaf': 12, 'min_samples_split': 6, 'n_estimators': 18}
Best criteria: gini
Best depth: 50
Best estimator: 18
```

As above, use K-Folds cross-validator to split data in train/test sets. Create a dictionary of hyperparameter cai and store hyper parameters and accuracy and tag using 'RandomForestClassifier'

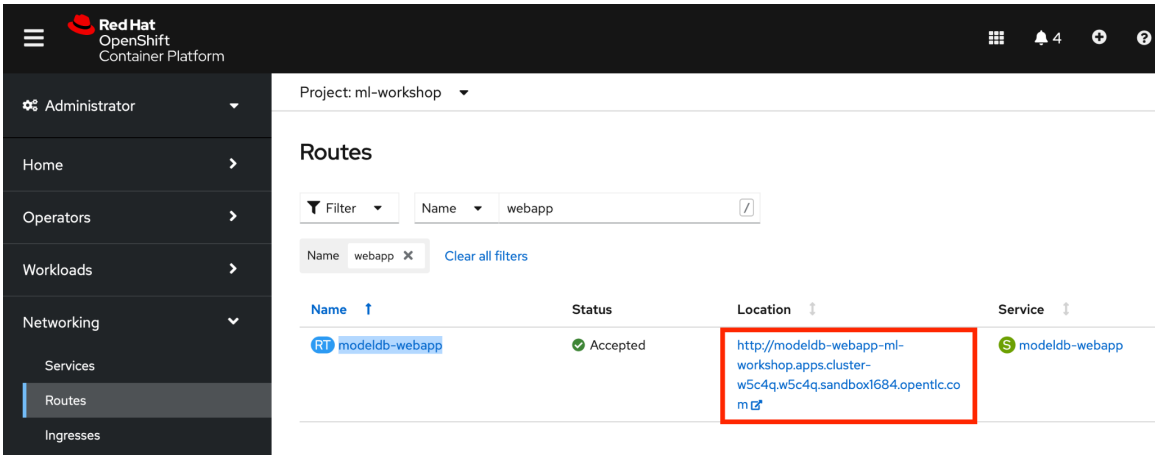
25
```
kfold = KFold(n_splits = 3)
hyperparameters = dict(max_depth=40, criterion='gini',min_samples_leaf = 12 ,min_samples_split = 8, n_estimators = 22)
model = RandomForestClassifier(max_depth=40, criterion='gini',min_samples_leaf = 12 ,min_samples_split = 8, n_estimators = 2
model = model.fit(X_train, y_train)
joblib.dump(model, 'rft.pkl')
results = model_selection.cross_val_score(model,X,Y,cv = kfold)
print(results)
print('Accuracy',results.mean()*100)
store = get_meta_store()
store.log_hyperparameters(hyperparameters)
store.log_model(model)
store.log_metric('Accuracy',results.mean()*100)
store.log_tag("RandomForestClassifier")
store.log_attribute("data_file_location", "data/full_data_csv/a.csv")
store.log_attribute("data_file_version", data_file_version)
```

23. **You need to modify this cell before you execute it.**
    Modify the 3rd line substituting your username in place of user29.
    Then we create a Random Forest Classifier with these hyper parameters.
24. This cell and cell 25 are equivalents of the previous Decision Tree classifier cells.

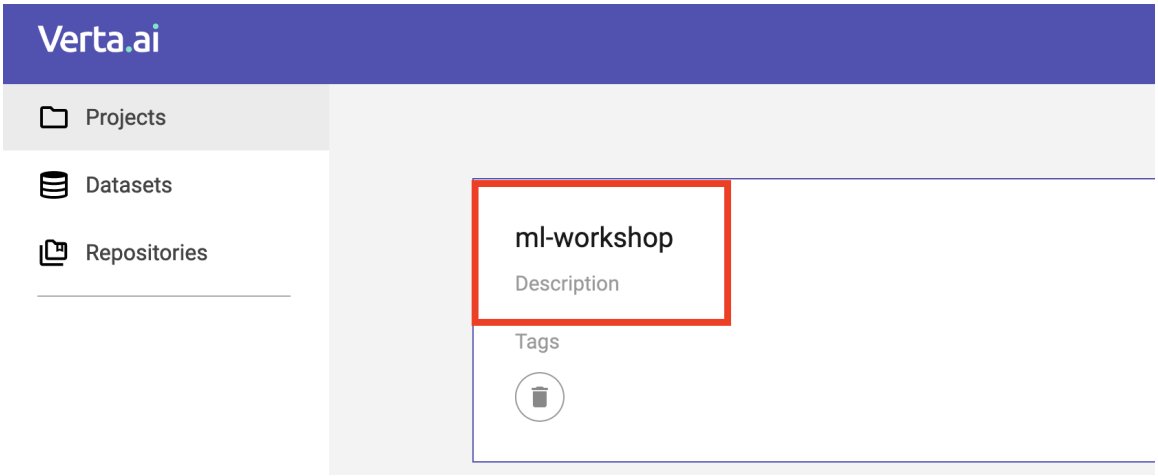
Part 3: Visualise the Model.

Let's use Verta to visually compare the model performance.

1. Open the OpenShift console tab in your browser.
2. Select the **Administrator Perspective**.
3. Click **Networking > Routes**.
4. Type ***webapp*** in the Filter text box.
   OpenShift will display the link to the Verta tool.

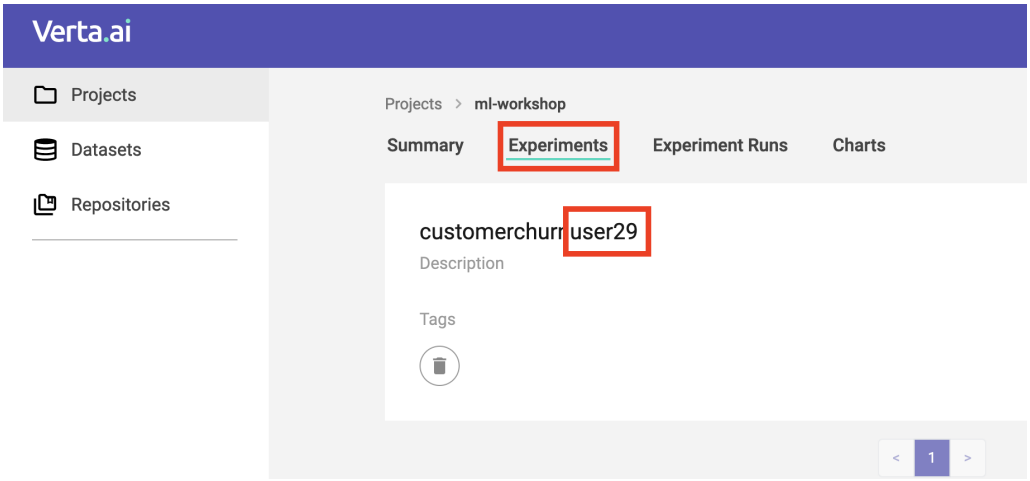


5. Click the hyperlink in the **Location** column
   OpenShift will launch the Verta console in a new browser tab.
6. Click the **ml-workshop** project tile.
   Verta will display the project details.
   This project was created when you ran your experiments and contains all of the results and parameters used to tune the models.

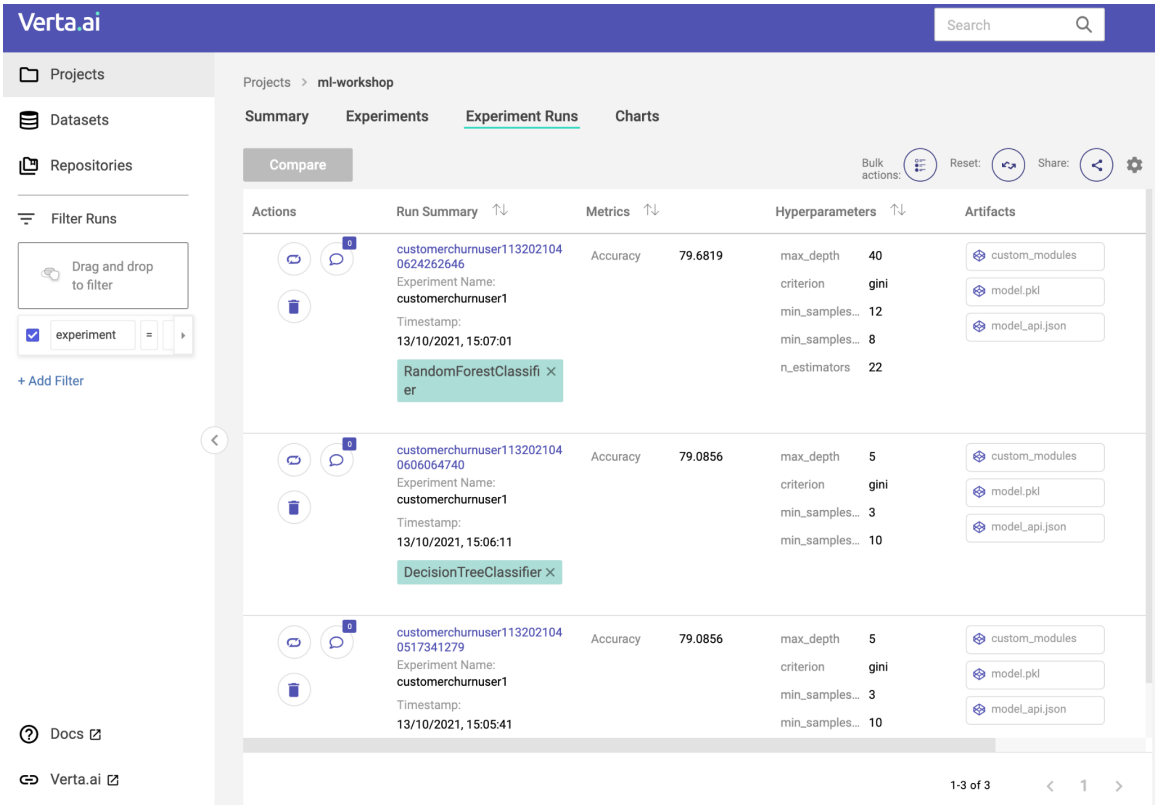


7. Click the **Experiments** menubar

   Verta displays all the experiments you have run to date. The experiments are grouped by your username (e.g. customerchurn**user29**. This is the result of your text substitution in the previous section, where set your experiment names to be customerchurn**userXX.**)

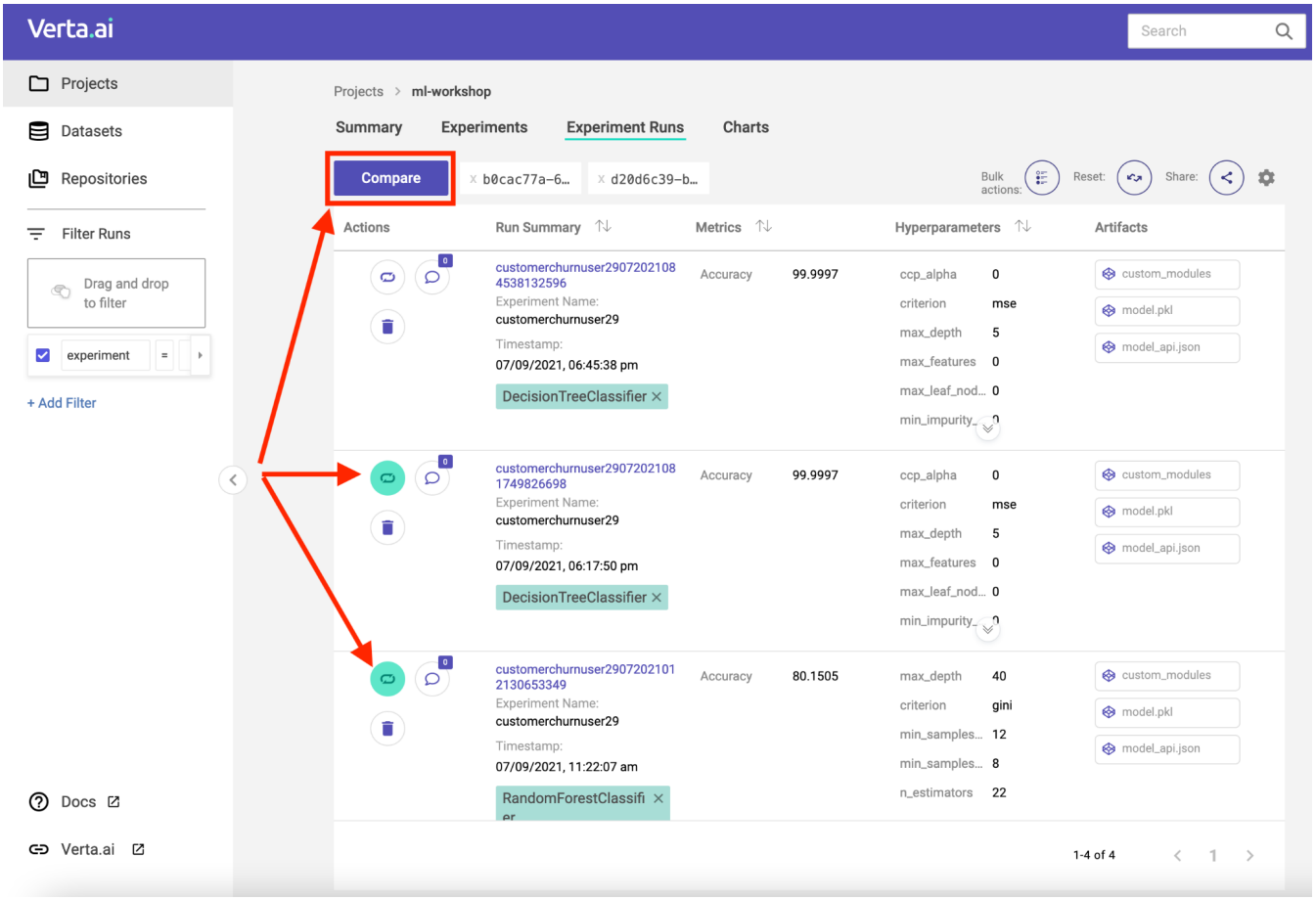


8. Click on the tile containing your username.

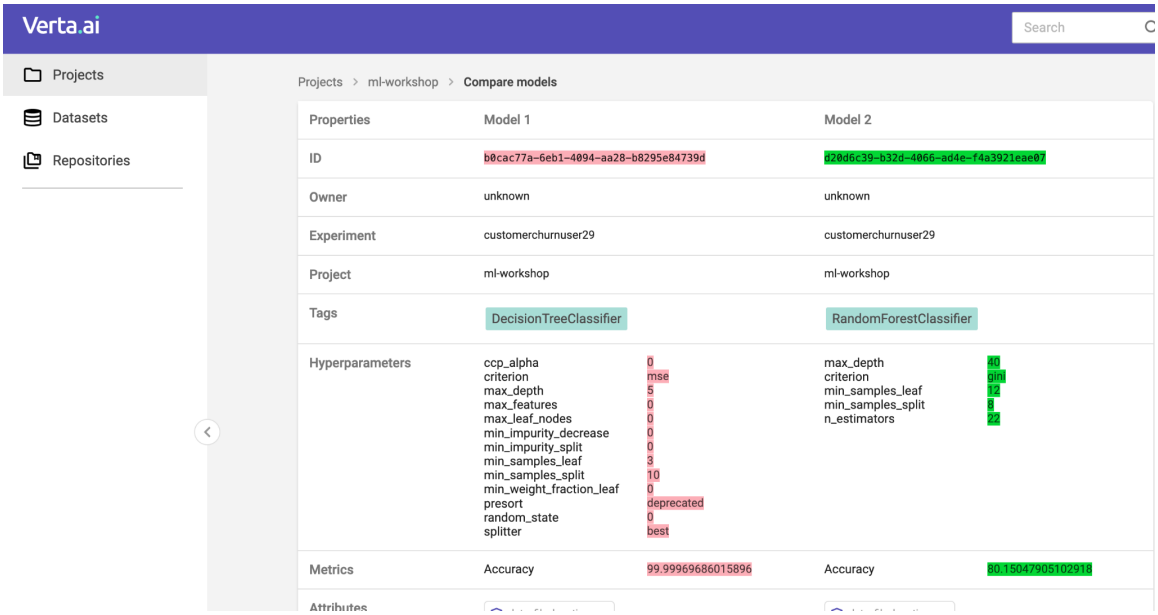


Verta displays all of the data that was sent to Verta using the ***store*** method in the notebook you ran earlier. Observe the: Parameters, Hyper parameters, Accuracies, Binaries etc.

9. Click the green looped arrows against any 2 experiments, the compare method becomes available.

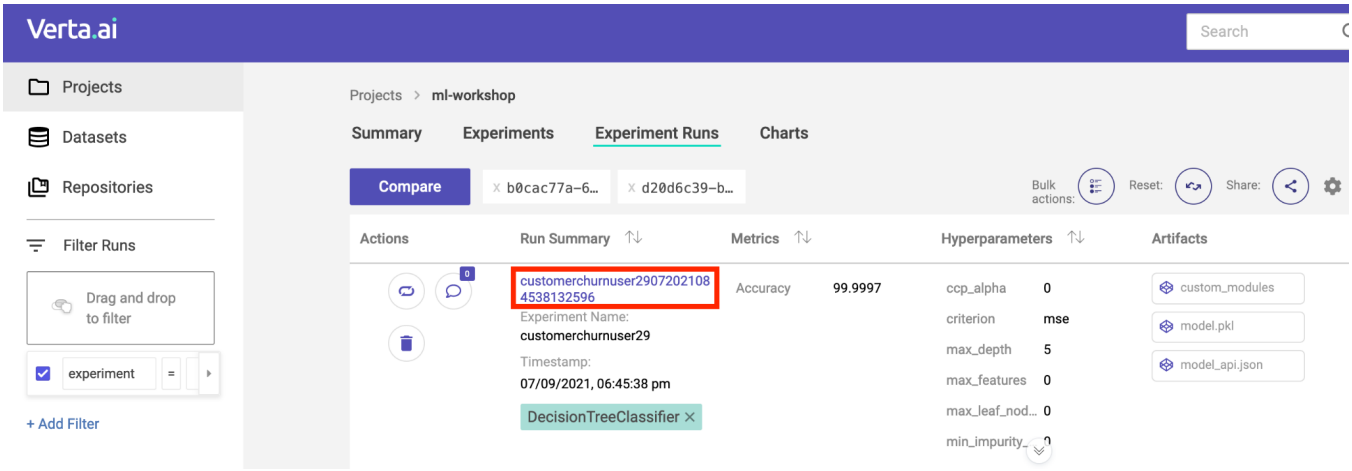


Verta displays the differences side-by-side.



10. Click the browser **Back** button to return to the previous screen.

    Observe that every experiment gets allocated an Experiment Id that contains a timestamp and the user's username. This ensures that the ID is unique across our team. We will also use this id when we push our chosen model to production using the ML OPs pipeline later.

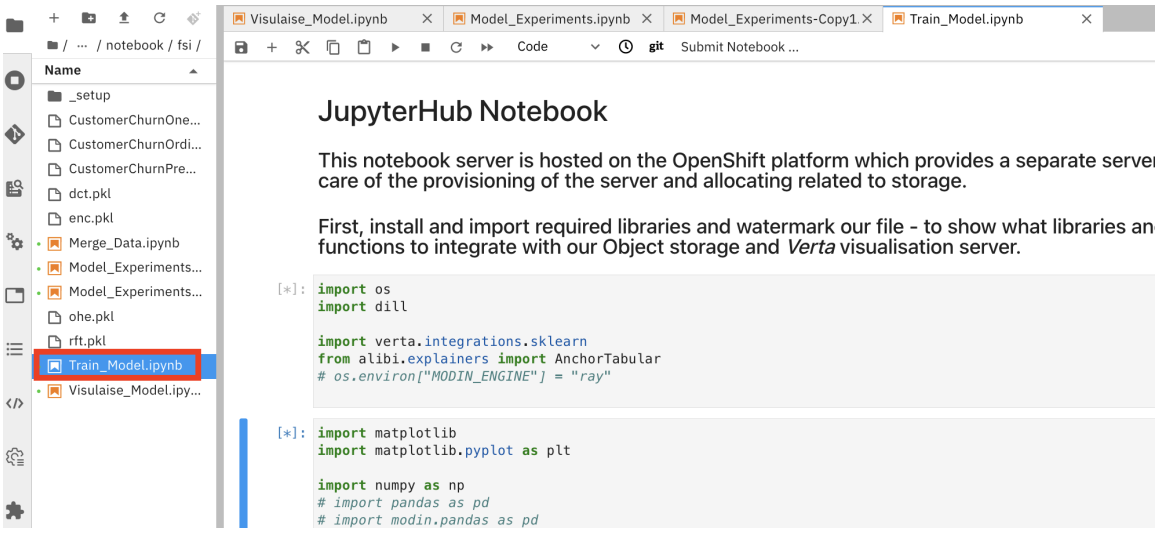


Spend the next few minutes exploring Verta and its capabilities before moving to the next lab.

Part 4: Train the Model

Now following examination of our experiments In Verta, let's assume for performance reasons, as a data scientist, you've decided to proceed with the Decision Tree Classifier experiment and push that model to production.

1. Open the Train_Model.ipynb notebook in Jupyter notebook



You have already encountered most of the cells here in the 2 previous notebooks. Therefore, we will only deal with the cells you need to change and the new cells.

You'll need to change each of these 2 cells, cell 4 and cell 5:

- Cell 4:
  As previously, replace the third line replacing user29 with your userid. Save the file
  Next we connect to Minio to pull our raw data from.
  Also connect to Verta to push our experiment data to. Set project within Verta to be ml-workshop.
- Cell 5:
  Here again, we retrieve the CSV we prepared in the Data Engineer lab earlier. To do that replace the string that's underlined in the screenshot with the string representing your file, **YOUR_CSV_FILE** from earlier in this lab. Save the file and continue executing.
  In this cell we also output the first 5 lines of the file - so the data scientist can get a quick view of the data.

[4]:
```
dateTimeObj = datetime.now()
timestampStr = dateTimeObj.strftime("%d%Y%H%M%S%f")
experiment_name = "customerchurnuser29"
experiment_id = experiment_name + timestampStr


def get_s3_server():
    minioClient = Minio('minio-ml-workshop:9000',
                    access_key='minio',
                    secret_key='minio123',
                    secure=False)

    return minioClient

def get_verta():
    client = Client("http://172.30.46.176:3000")
    return client

def get_meta_store():
    client = get_verta()
    proj = client.set_project("ml-workshop")
    client.set_experiment(experiment_name)
    run = client.set_experiment_run(experiment_id)
    return run
```

In this next section, on the second line, insert the value you retrieved from Minio object storage earlier - represer
name of your csv file in Minio. This is the file pushed by the data engineer in the format: full_data_csv{USERNAM

In my case this value is: full_data_csvuser29/part-00000-59149e08-583c-46a5-bfa0-0b3abecbf1a3-c000.csv (yours will be diffe

We refer to this fully qualified name in the Github instructions as CSV-FILE

[5]:
```
minioClient = get_s3_server()
data_file = minioClient.fget_object("data", "full_data_csvuser29/part-00000-8a222f86-81cd-49e5-bc60-e28e3ac60d65-c000.csv", "/t
data_file_version = data_file.version_id
data = pd.read_csv('/tmp/data.csv')
data.head(5)
```

2. Click **File > Save Notebook** to save your work.
3. Click in cell **[1]** of the notebook.
4. Step through the notebook as before by pressing **[Shift] + [Return]**.

As we've chosen to proceed with Decision Tree Classifier experiment, notice we use the same hyper parameters etc here as we did previously:

[14]:
```
from sklearn.tree import DecisionTreeRegressor

def train_and_save_model():
    kfold = KFold(n_splits = 3)
    model = DecisionTreeRegressor(max_depth=5, criterion='mse',min_samples_leaf = 3 ,min_samples_split = 10)
    store = get_meta_store()
    model = model.fit(X_train, y_train, run=store)
    joblib.dump(model, 'CustomerChurnPredictor.sav')
    results = model_selection.cross_val_score(model,X,Y,cv = kfold)
    print(results)
    print('Accuracy',results.mean()*100)

    store.log_model(model, overwrite=True)
    store.log_metric('Accuracy',results.mean()*100)
    store.log_tag("DecisionTreeClassifier")
    store.log_attribute("data_file_location", "data/full_data_csv/a.csv")
    store.log_attribute("data_file_version", data_file_version)

    return model
```

Notice here, we create an *Explainer* object, which we could use to deploy later.

[15]:
```
from alibi.utils.data import gen_category_map

def explain_model(model, X_train, X_test_record):
    fnames = X_train.columns.tolist()
    predict_fn = lambda x: model.predict_proba(x)
    explainer = AnchorTabular(predict_fn, fnames)
    explainer = explainer.fit(X_train.values, disc_perc=[25, 50, 75])
    explanation = explainer.explain(X_test_record.values[0])
    print('Anchor: %s' % explanation['anchor'])
    print('Precision: %.2f' % explanation['precision'])
    print('Coverage: %.2f' % explanation['coverage'])
    return explainer
```

(Due to time constraints, we don't do that in today's hands-on-workshop)

Finally of note, in this notebook we create binaries for

- The model,
- the 2 encoders, Ordinal and One Hot encoders
- The explainer

In this cell we use our minioClient, to push these binaries to the models S3 bucket, in a folder named with your experimentId (according to your username, the substitution you just made in cell 4).

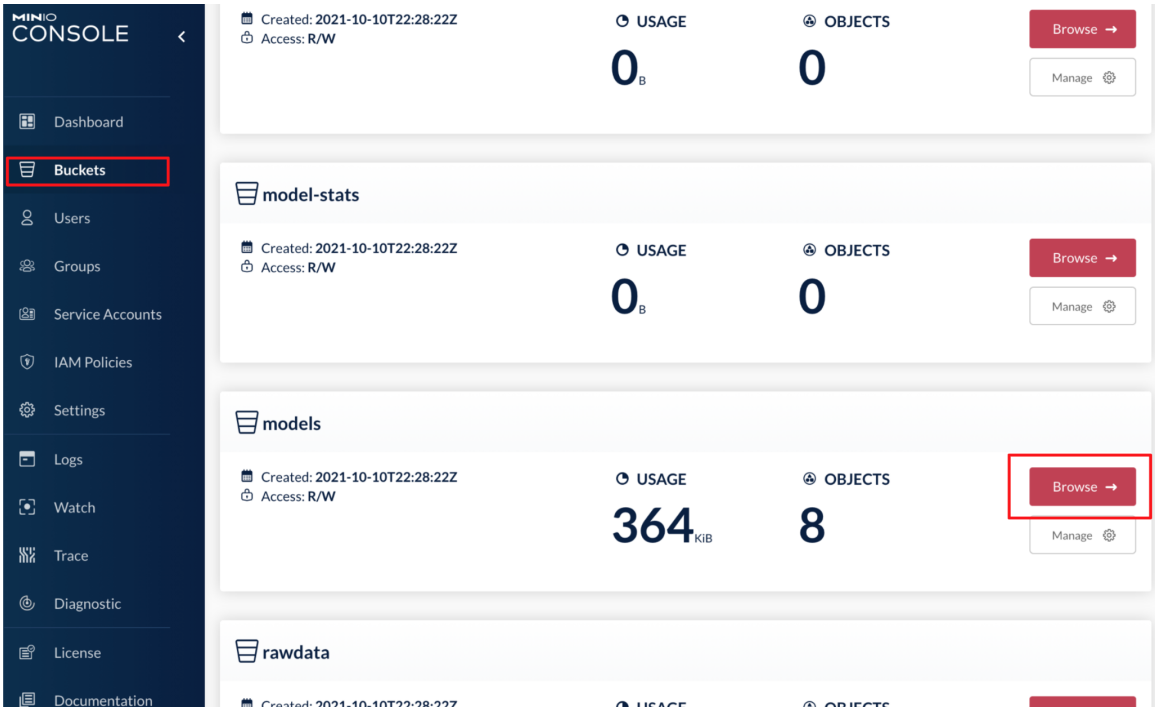
```
minioClient = get_s3_server()
minioClient.fput_object(bucket_name='models', object_name=experiment_id  +'/CustomerChurnPredictor.sav' , file_path='./CustomerChurnPredictor.sav')
minioClient.fput_object(bucket_name='models', object_name=experiment_id  +'/CustomerChurnPredictorAlibi.dill' , file_path='./CustomerChurnPredictorAlibi.dill')
minioClient.fput_object(bucket_name='models', object_name=experiment_id  +'/CustomerChurnOrdinalEncoder.pkl' , file_path='./CustomerChurnOrdinalEncoder.pkl')
minioClient.fput_object(bucket_name='models', object_name=experiment_id  +'/CustomerChurnOneHotEncoder.pkl' , file_path='./CustomerChurnOneHotEncoder.pkl')
```

Part 6: View the Model Binaries

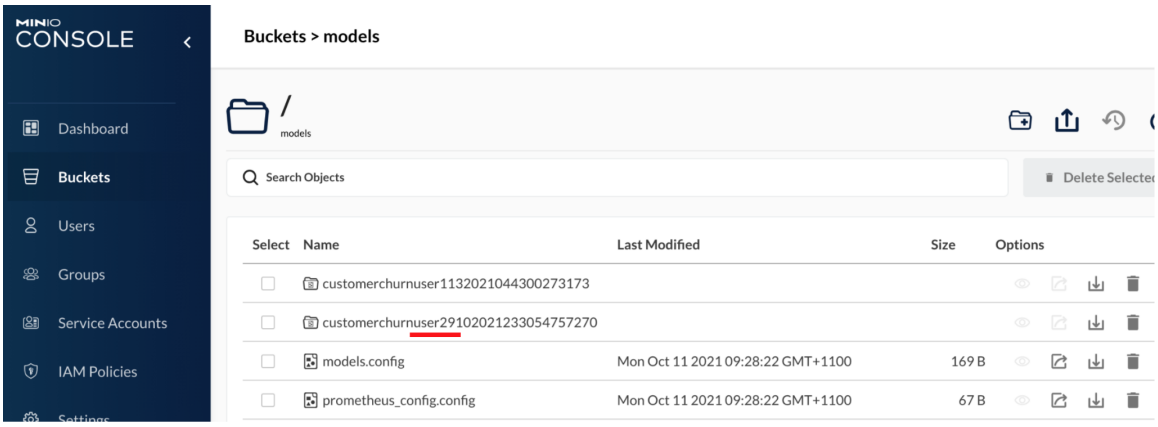
In the previous section we trained the model and stored the trained model in S3 object storage (implemented using the Minio tool, though the same means could be used to push them to an enterprise storage solution, such as Ceph).

Open the Minio browser tab from the earlier exercise. (Refer to Instructions to access your prepared data file from the previous lab if you have closed it.)

1. Click **Buckets**
   Minio displays all of the S3 storage buckets.
   Locate the **models** bucket
2. Click **Browse**.

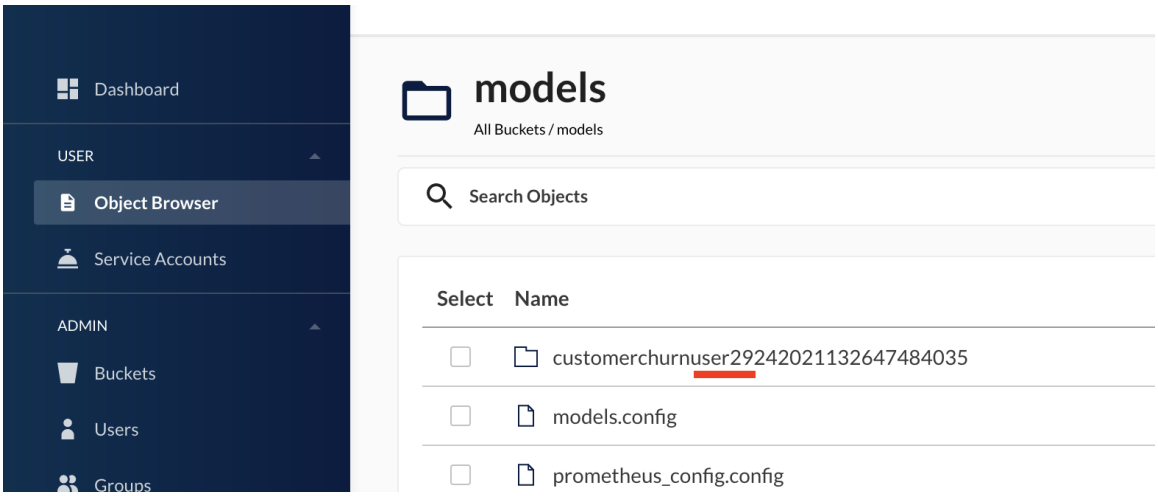


Minio displays the contents of the bucket. Within this bucket you will see a folder for each person participating in this lab. Locate the folder with your username. E.g. user29

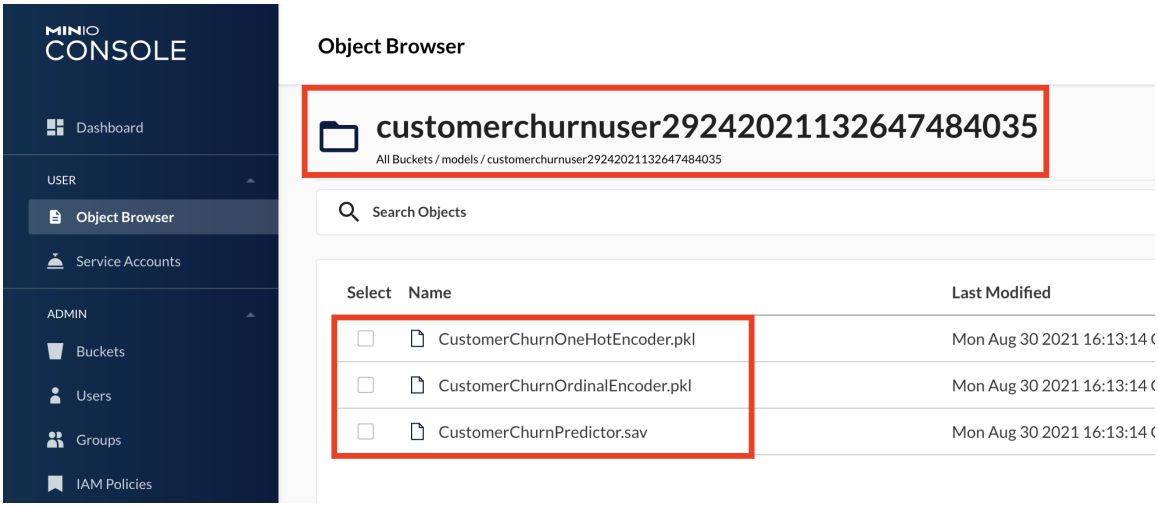


Find the folder containing your username - this is mine for user29. Click into yours.

Pay careful attention to ensure you choose the correct folder name.



3. Click your folder
   Minio displays the model files.
4. Copy the model folder name to the clipboard and paste it in the editor you used earlier for your CSV file. E.g. *customerchurnuser29242021132647484035*.
   You will use this when you deploy the model later.


MINIO
CONSOLE
Dashboard
USER
Object Browser
Service Accounts
ADMIN
Buckets
Users
Groups
IAM Policies
Object Browser
customerchurnuser29242021132647484035
All Buckets / models / customerchurnuser29242021132647484035
Search Objects
Select
Name
Last Modified
CustomerChurnOneHotEncoder.pkl
Mon Aug 30 2021 16:13:14
CustomerChurnOrdinalEncoder.pkl
Mon Aug 30 2021 16:13:14
CustomerChurnPredictor.sav
Mon Aug 30 2021 16:13:14