

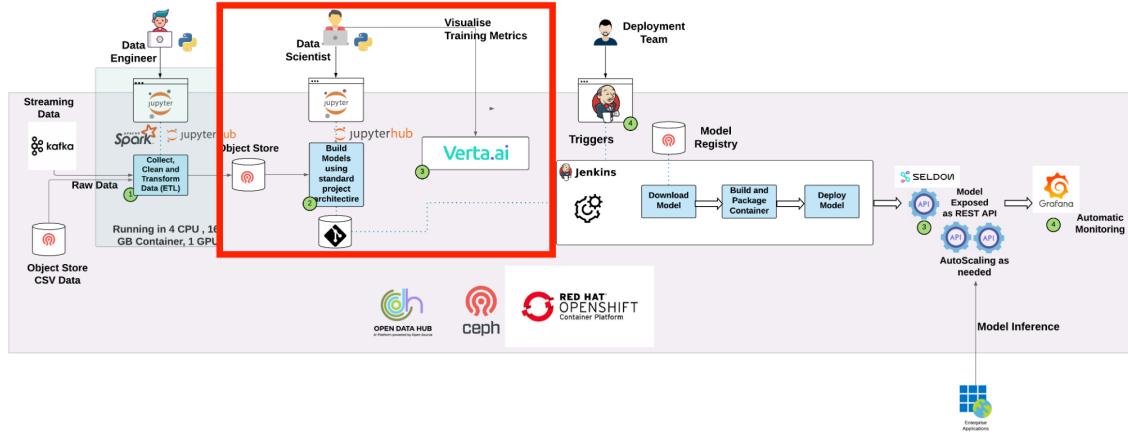
Lab 2 - Data Science

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 dropdown , navigate to Network -> Routes. Filter on minio - and open the *minio-ml-workshop-ui* route as shown.

The screenshot shows the OpenShift Admin UI. The left sidebar is titled "Administrator" and includes "Home", "Operators", "Workloads", "Networking" (with "Services" and "Routes" selected), "Ingresses", and "NetworkPolicies". The main area is titled "Routes" and has a filter bar with "Name" set to "minio". It lists two routes:

Name	Status	Location	Service
minio-ml-workshop	Accepted	http://minio-ml-workshop-ml-workshop.apps.cluster-fcla.fcla.sandbox640.opentlc.com	minio-ml-workshop
minio-ml-workshop-ui	Accepted	http://minio-ml-workshop-ui-ml-workshop.apps.cluster-fcla.fcla.sandbox640.opentlc.com	minio-ml-workshop

Enter the username and password minio / minio123

Choose Object Browser on the left and click into the data bucket as shown:

The screenshot shows the MinIO Console. The left sidebar has sections for "MINIO CONSOLE", "USER" (with "Object Browser" selected), "ADMIN" (with "Buckets", "Users", "Groups", "IAM Policies", and "Settings"), and "Dashboard". The right panel is titled "Object Browser" and shows the "All Buckets" view. A search bar at the top says "Filter Buckets". Below it is a table with columns "Name" and "Used Space". The "data" bucket is highlighted with a red box.

Name	Used Space
data	726 KiB
model-stats	0 B
models	32 KiB
queryc1	154 KiB
rawdata	758 KiB



Click into the folder **full_data_csvuserXX** that contains **your username**, e.g. this is the folder for user29:

The screenshot shows the Minio Object Browser interface. On the left, there's a sidebar with 'MINIO CONSOLE' at the top, followed by 'Dashboard', 'USER' (with 'Object Browser' selected), 'Service Accounts', 'ADMIN', and 'Buckets'. The main area is titled 'Object Browser' and shows a list of objects under the 'data' bucket. A search bar at the top says 'Search Objects'. Below it is a table with columns: 'Select', 'Name', 'Last Modified', 'Size', and 'Options'. One item is listed: 'full_data_csvuser29'. The 'full_data_csvuser29' folder is highlighted with a red box.

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

Click on the long filename:

This screenshot shows the 'full_data_csvuser29' folder selected in the Minio Object Browser. The table below lists two objects: '_SUCCESS' and 'part-00000-8e4fbad4-0fc3-42c6-940c-612515753086-c000.csv'. The 'part-00000-8e4fbad4-0fc3-42c6-940c-612515753086-c000.csv' file is highlighted with a red box.

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:

The screenshot shows the 'part-00000-8e4fbad4-0fc3-42c6-940c-612515753086-c000.csv' file selected in the Minio Object Browser. The file path 'full_data_csvuser29/part-00000-8e4fbad4-0fc3-42c6-940c-612515753086-c000.csv' is highlighted with a red box.

Paste this into a text editor of your choice - I use Sublime Text. Delete the two embedded spaces on either side of the forward slash:



Your file should now look like 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.

Instructions for the first Data Science workshop - Visualisation

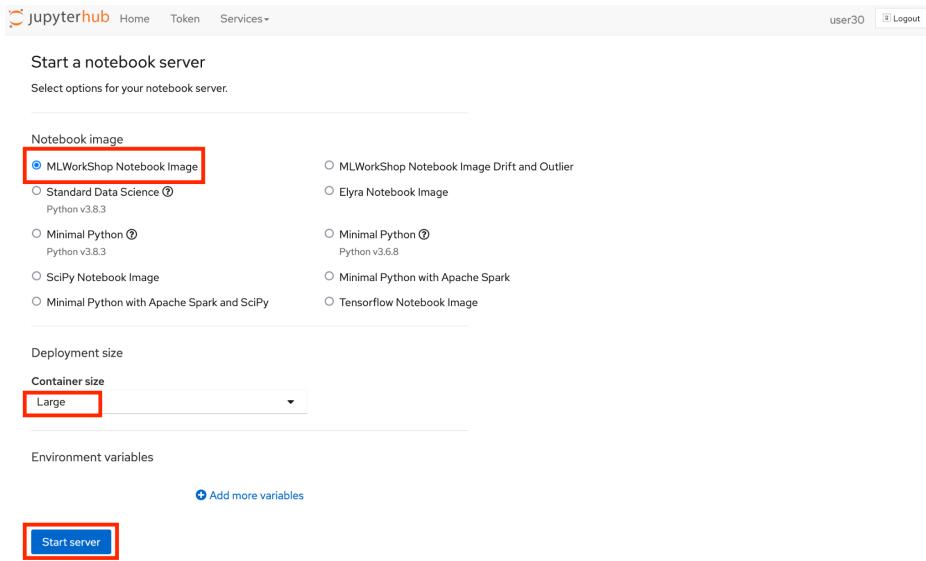
Now to our Data Science focused Jupyter notebooks.

Choose the Administration dropdown , navigate to Network -> Routes. Filter on *Jupyterhub* - and open the route.

Name	Status	Location	Service
jupyterhub	Accepted	https://jupyterhub-ml-workshop.apps.cluster-fcla.fcla.sandbox640.opentlc.com	jupyterhub

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.



Start a notebook server

Select options for your notebook server.

Notebook image

- MLWorkShop Notebook Image
- Standard Data Science ⓘ
Python v3.8.3
- Minimal Python ⓘ
Python v3.8.3
- SciPy Notebook Image
- Minimal Python with Apache Spark and SciPy
- MLWorkShop Notebook Image Drift and Outlier
- Elyra Notebook Image
- Minimal Python ⓘ
Python v3.6.8
- Minimal Python with Apache Spark
- Tensorflow Notebook Image

Deployment size

Container size

Large

Environment variables

Add more variables

Start server

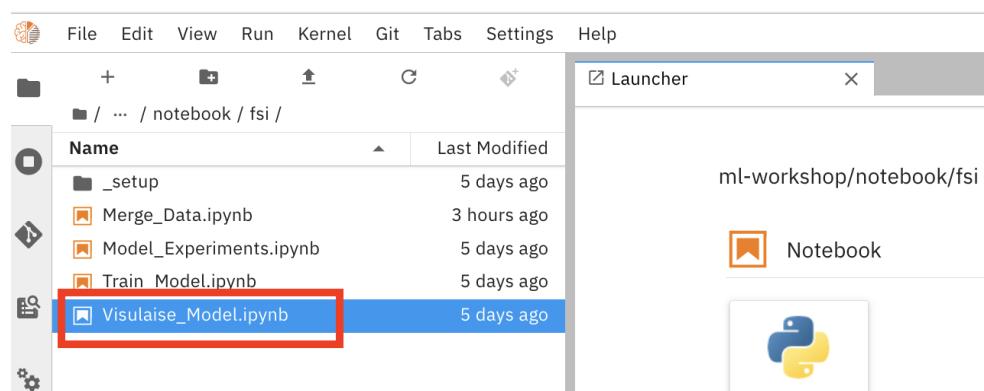
This is a custom image our OpenShift administrator has created - containing the exact tools we as a team or organisation require - optionally with the desired versions of each of the tools it contains. This is another powerful feature, allowing a consistent toolset to be used - eliminating any version mismatches we often see when practitioners use different tool versions.

Click *Start Server*. A few moments later the *files* view appears. You'll notice

- a different GUI look and feel - 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>

Again drill into *ml-workshop* → *notebook*. Depending on your track **telco** or **fsi**, choose the appropriate subfolder (in this example **fsi**).

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):





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.

The screenshot shows a Jupyter Notebook interface with a sidebar containing file navigation and a central workspace. The workspace displays three code cells:

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

        import numpy as np
        import pandas as pd

        import watermark
# import s3fs
        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 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
```

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

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

        return minioClient
```

1. Import our desired Python libraries. (notice we don't need to do any **pip installs** - 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()
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	No
1	463	Male	0	Yes	Yes	4	Yes	Yes	Branch	No	...	Yes	No	No
2	471	Female	1	No	No	17	Yes	No	No	No	Not Available	...	Not Available	Not Available
3	496	Male	0	No	No	22	No	Not available	Mobile	No	...	Yes	No	No
4	833	Female	0	Yes	Yes	70	Yes	No	Mobile	Yes	...	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. 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.

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.

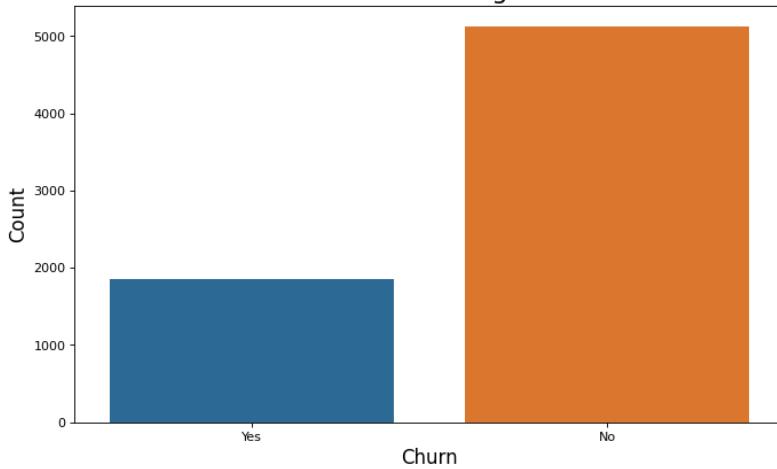
8 [8]: `data.isnull().sum()`

```
[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')`

Distribution of the Target Variable



10 [0]: `# 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]: 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)

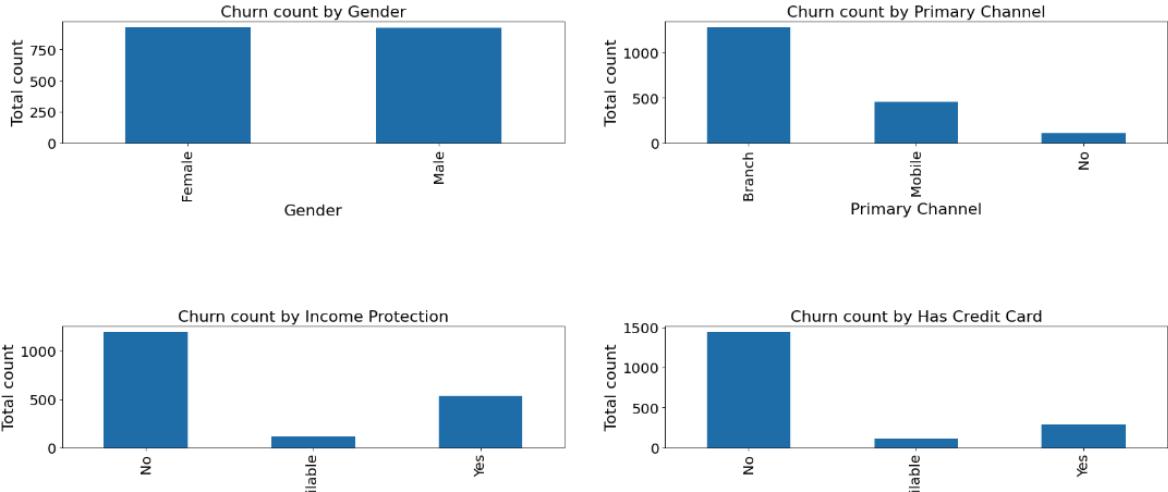
data.groupby('IncomeProtection').Churn.sum().plot(kind='bar', ax=ax3)
ax3.set_ylabel('Total count',fontsize = 20)
ax3.set_xlabel('Income Protection',fontsize = 20)
ax3.tick_params(labelsize = 18)
ax3.set_title('Churn count by Income Protection',fontsize = 20)

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)

```



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



Instructions for second Data Science workshop - 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.

Open up Model_Experiments.ipynb and begin executing as before using SHIFT+RETURN to proceed through the notebook. Follow the descriptions of each cell.

Note there are 3 changes you'll need to make to your cells - which we'll highlight below:



```
[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 store name of your csv file in Minio. This is the file pushed by the data engineer in the format: full_data_csvuser29/part-00000-59149e08-583c-46a5-bfa0-0b3abecbf1a3-c

In my case this value is: full_data_csvuser29/part-00000-8a222f86-81cd-49e5-bc60-e

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 x 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 x 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 x 46 columns
```

17

```
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_split = min_samples_split)

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 candidates for DecisionTreeClassifier, assess results, print and store hyper parameters and accuracy and tag using `store`

```
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, 'dtc.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 user name. Your username is user1:

```
experiment_name = "customerchurnuser1"
```

19. Create a DecisionTreeClassifier with these hyper parameters
20. Use 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 pushes 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_leaf)
# 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 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 = 22)
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.

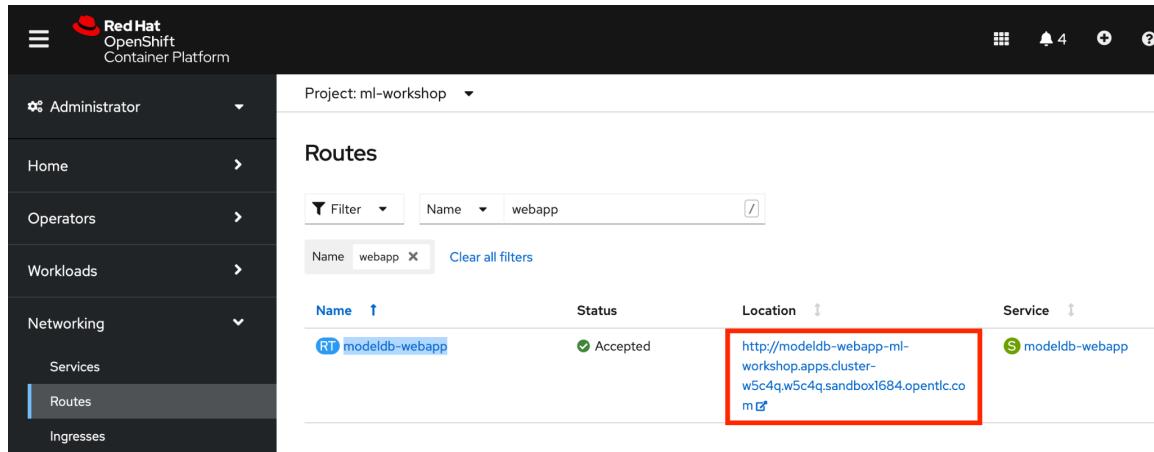
Modify the 3rd line substituting your username in place of user29.

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.

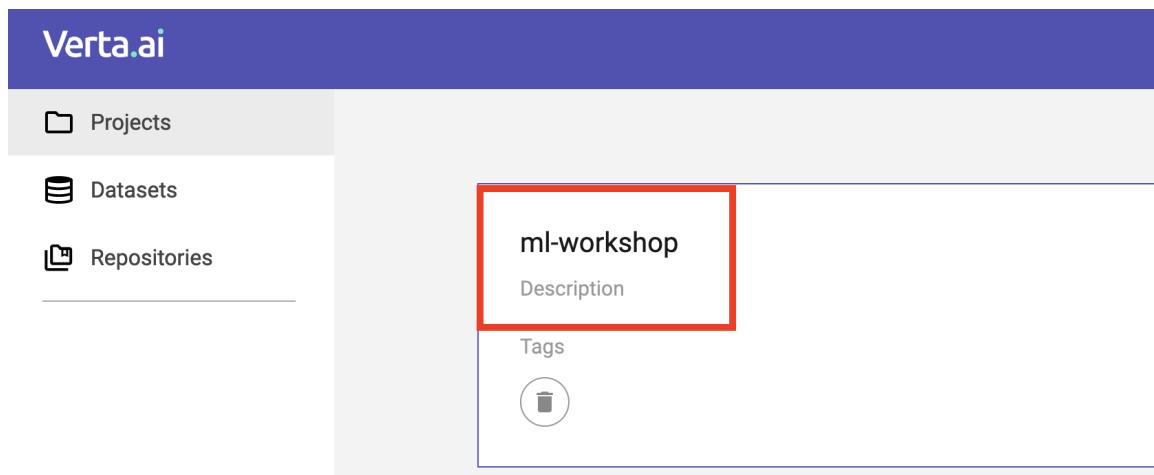
Instructions to access your model visualisation and repository Verta.

In OpenShift choose the Administration dropdown , navigate to Network -> Routes. Filter on *webapp* - and open the *modeldb-webapp* route, which is our Verta tool, as shown. You won't need to supply any credentials.



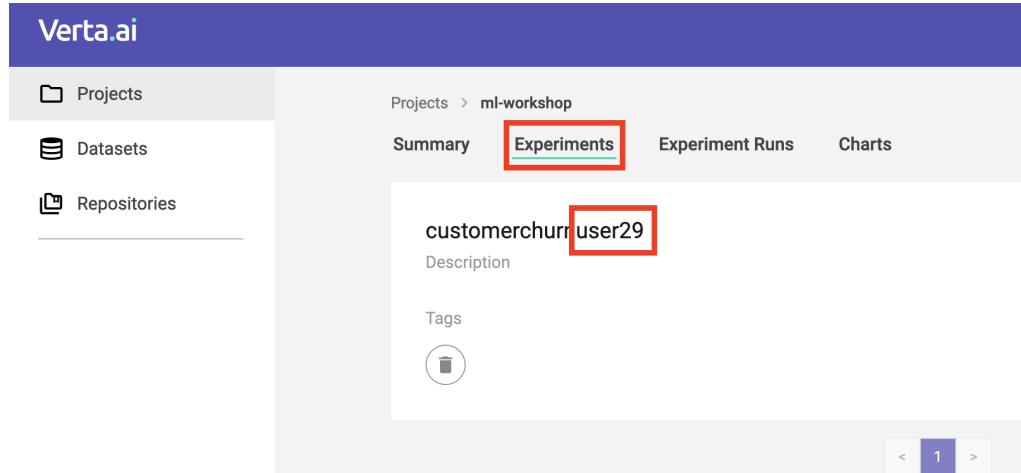
The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar has a navigation menu with 'Administrator' at the top, followed by 'Home', 'Operators', 'Workloads', 'Networking' (with 'Services' and 'Routes' under it), and 'Ingresses'. The 'Routes' item is currently selected and highlighted with a blue background. The main content area is titled 'Routes' and shows a table with one row. The table columns are 'Name', 'Status', 'Location', and 'Service'. The single row contains 'modeldb-webapp' in the Name column, 'Accepted' in the Status column, a location URL in the Location column, and 'modeldb-webapp' in the Service column. The URL in the Location column is <http://modeldb-webapp-ml-workshop.apps.cluster-w5c4q.w5c4q.sandbox1684.opentlc.com>, which is also highlighted with a red box.

Drill into *ml-workshop*, this is the high level project name we used to push all of our experiments in the *Model_Experiments.ipynb* notebook.



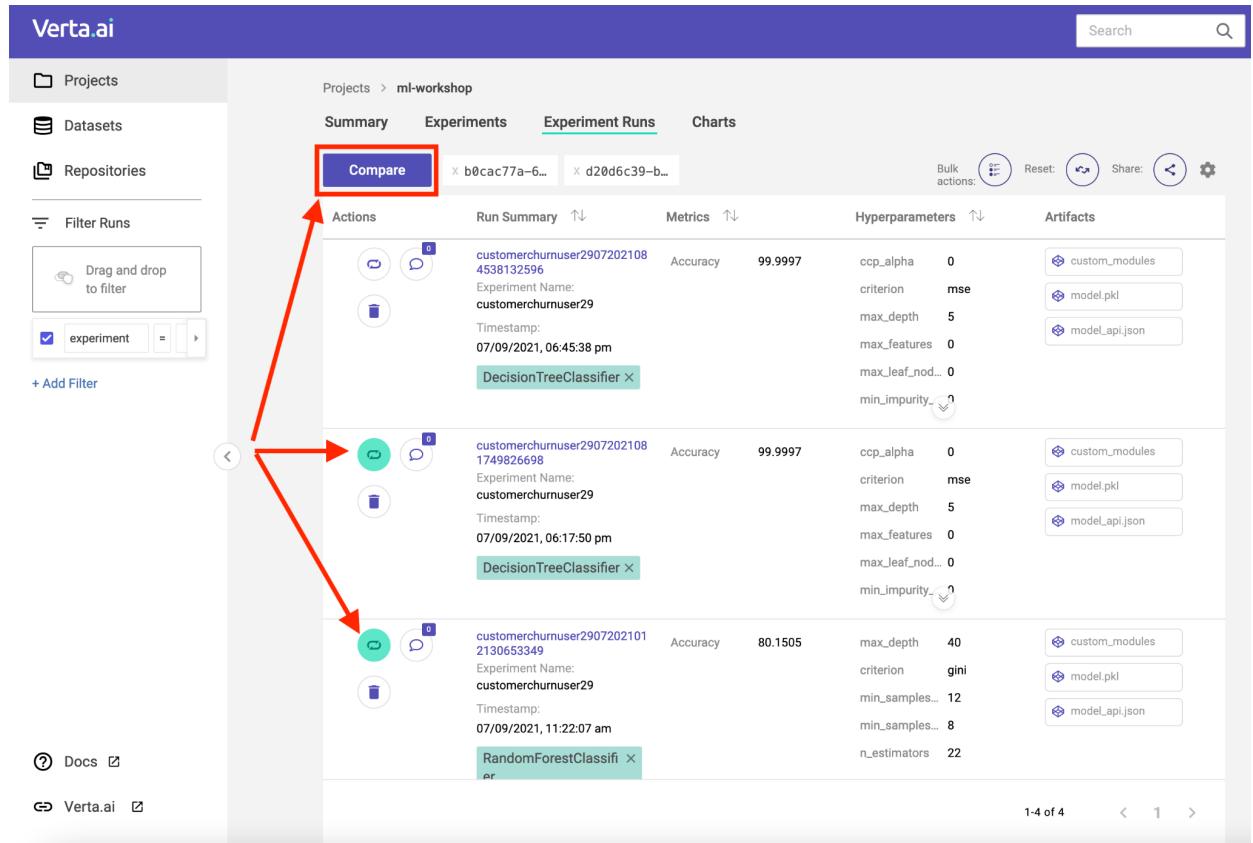
The screenshot shows the Verta.ai interface. The left sidebar has three items: 'Projects', 'Datasets', and 'Repositories'. The 'Projects' item is selected and highlighted with a blue background. The main content area shows a project card for 'ml-workshop'. The card has a title 'ml-workshop', a 'Description' section below it, and a 'Tags' section with a trash icon. A red box highlights the 'ml-workshop' title.

Choose *Experiments* and notice all experiments are grouped by your username, in my case customerchurn**user29**. This is the result of your text substitution in the previous section, where set your experiment names to be customerchurn**userXX**.



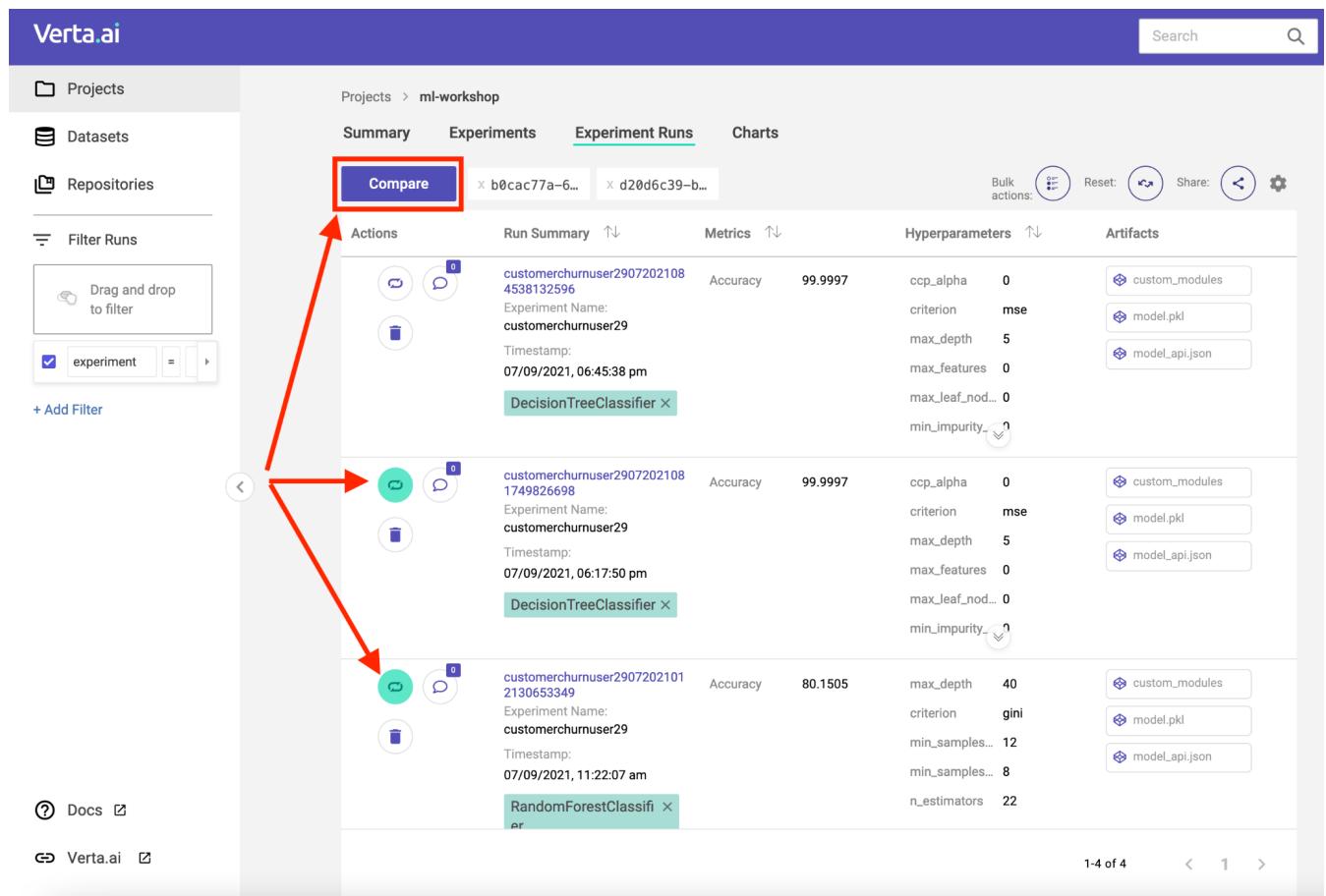
The screenshot shows the Verta.ai web interface. On the left, there's a sidebar with 'Projects', 'Datasets', and 'Repositories'. The main area shows 'Projects > ml-workshop'. Below that, there are tabs for 'Summary', 'Experiments' (which is selected and highlighted with a red box), 'Experiment Runs', and 'Charts'. A search bar at the top right has 'Search' and a magnifying glass icon. The main content area displays an experiment named 'customerchurnuser29', which is also highlighted with a red box. It includes sections for 'Description' and 'Tags', with a trash can icon.

Find your folder containing your username. Drill into that folder:



This screenshot shows the 'Experiment Runs' tab for the 'ml-workshop' project. The left sidebar is identical to the previous screenshot. The main area shows the 'Experiment Runs' tab selected. At the top, there's a 'Compare' button highlighted with a red box, and two other experiment runs are listed: 'b0cac77a-6...' and 'd20d6c39-b...'. Below these, another experiment run is listed: '2130653349'. Each run has a detailed card showing its experiment name ('customerchurnuser29'), accuracy (99.9997 or 80.1505), hyperparameters (like 'ccp_alpha', 'criterion', etc.), and artifacts (like 'custom_modules', 'model.pkl', 'model_api.json'). Red arrows point from the 'Compare' button to each of the three experiment runs listed below it.

Notice all of the data we sent to Verta using the **store** method earlier are here in Verta. Parameters, Hyper parameters, Accuracies, Binaries etc. Notice if we click the green looped arrows against any 2 experiments, the compare method becomes available. Do that and click it.



The screenshot shows the Verta.ai interface for managing machine learning experiments. On the left, there's a sidebar with 'Projects', 'Datasets', 'Repositories', and a 'Filter Runs' section. The main area is titled 'ml-workshop' and shows three experiment runs:

Experiment Name	Accuracy	Hyperparameters	Artifacts
customerchurnuser2907202108 4538132596	99.9997	ccp_alpha: 0, criterion: mse, max_depth: 5, max_features: 0, max_leaf_nod... 0, min_impurity_< 0	custom_modules, model.pkl, model_api.json
customerchurnuser2907202108 1749826698	99.9997	ccp_alpha: 0, criterion: mse, max_depth: 5, max_features: 0, max_leaf_nod... 0, min_impurity_< 0	custom_modules, model.pkl, model_api.json
customerchurnuser2907202101 2130653349	80.1505	max_depth: 40, criterion: gini, min_samples_> 12, min_samples_< 8, n_estimators: 22	custom_modules, model.pkl, model_api.json

At the top, there are tabs for 'Summary', 'Experiments', 'Experiment Runs' (which is selected), and 'Charts'. A 'Compare' button is located in the top-left corner of the main content area. Red arrows highlight the 'Compare' button and the 'Compare' icons in the 'Actions' column of the experiment table.



Notice we can compare values side by side.

The screenshot shows the Verta.ai interface with the title "Compare models". On the left, there's a sidebar with "Projects", "Datasets", and "Repositories". The main area displays two models, "Model 1" and "Model 2", for comparison. The properties compared include:

	Model 1	Model 2
ID	b0cac77a-6eb1-4094-aa28-b8295e84739d	d20d6c39-b32d-4866-ad4e-f4a3321eaed7
Owner	unknown	unknown
Experiment	customerchurnuser29	customerchurnuser29
Project	ml-workshop	ml-workshop
Tags	DecisionTreeClassifier	RandomForestClassifier

Under "Hyperparameters", there is a table comparing various parameters for both models. The "Metrics" section shows Accuracy values of 99.99969686015896 for Model 1 and 0.15047805102918 for Model 2.

Return to the previous screen by clicking your browser's Back button.

Finally notice every experiment gets allocated an Experiment Id, containing 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.

The screenshot shows the "Experiment Runs" tab in the Verta.ai interface. The sidebar includes "Projects", "Datasets", "Repositories", and "Filter Runs" (with a "Drag and drop to filter" option). The main area shows a list of experiment runs with columns for "Actions", "Run Summary", "Metrics", "Hyperparameters", and "Artifacts". A specific run is highlighted with a red box, showing its details:

Actions	Run Summary	Metrics	Hyperparameters	Artifacts
	customerchurnuser2907202108 4538132596	Accuracy 99.9997	ccp_alpha 0 criterion mse max_depth 5 max_features 0 max_leaf_nodes 0 min_impurity_decrease 0 min_impurity_split 0 min_samples_leaf 3 min_samples_split 10 min_weight_fraction_leaf 0 presort deprecated random_state 0 splitter best	custom_modules model.pkl model_api.json

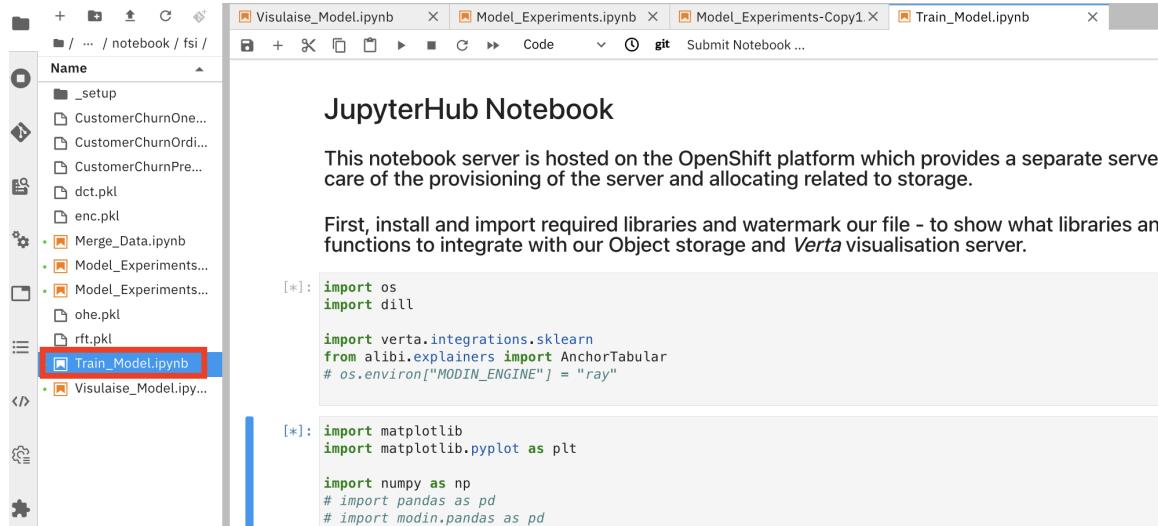
Below the table, it shows the Experiment Name: customerchurnuser29, the Timestamp: 07/09/2021, 06:45:38 pm, and the Model Type: DecisionTreeClassifier.

Feel free to browse around Verta, more than is shown than I described.

Instructions for the third Data Science step - Train Model

Now following examination of our experiments In Varta, 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.

We do that in the Train_Model notebook. Go ahead and open that.



The screenshot shows a JupyterHub Notebook interface. The left sidebar lists files and notebooks, with 'Train_Model.ipynb' highlighted. The main area displays the content of the 'Train_Model.ipynb' notebook. The first cell contains code to import os, dill, verta, and alibi.explainers. The second cell contains code to import matplotlib, numpy, and pandas.

```
[*]: import os
[*]: import dill
[*]: import verta.integrations.sklearn
[*]: from alibi.explainers import AnchorTabular
[*]: # os.environ["MODIN_ENGINE"] = "ray"
[*]: import matplotlib
[*]: import matplotlib.pyplot as plt
[*]: import numpy as np
[*]: # import pandas as pd
[*]: # import modin.pandas as pd
```

You've already encountered most of the cells here in the 2 previous notebooks. Therefore, I'll 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 - represent name of your csv file in Minio. This is the file pushed by the data engineer in the format: full_data_csv{USERNAME}.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

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

Save the file and work your way through the file, executing as you go by clicking 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).

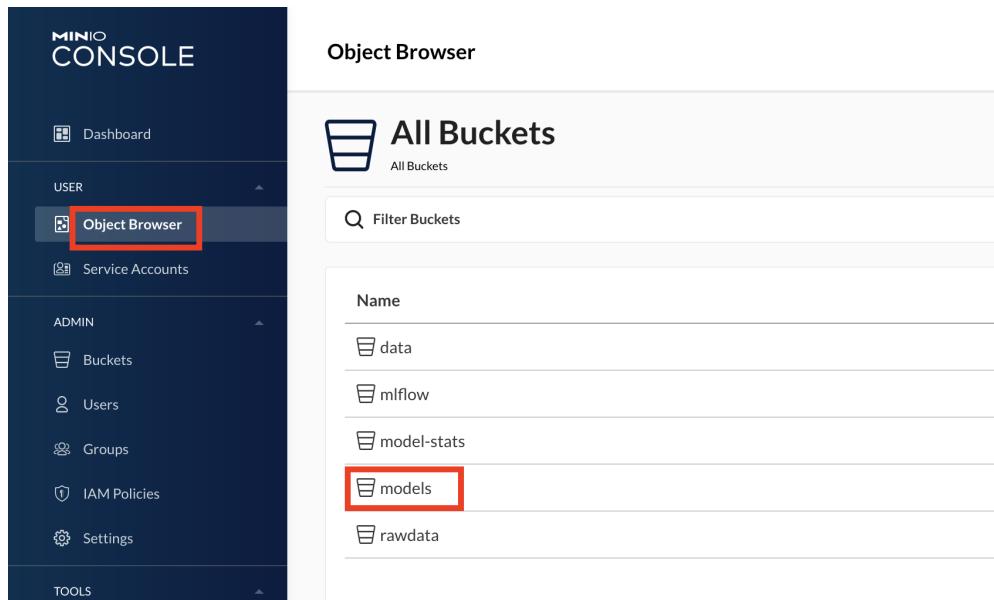
```
minioClient = get_s3_server()
minioClient.fput_object(bucket_name="models", object_name=experiment_id + '/CustomerChurnPredictor.sav' , file_path='./CustomerChurnPredictor.sav')
minioClient.fput_object(bucket_name="models", object_name=experiment_id + '/CustomerChurnPredictorAlibi.dill' , file_path='./CustomerChurnPredictorAlibi.dill')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

Instructions to view your model binaries in S3 object storage.

As just mentioned, we push our models to 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.

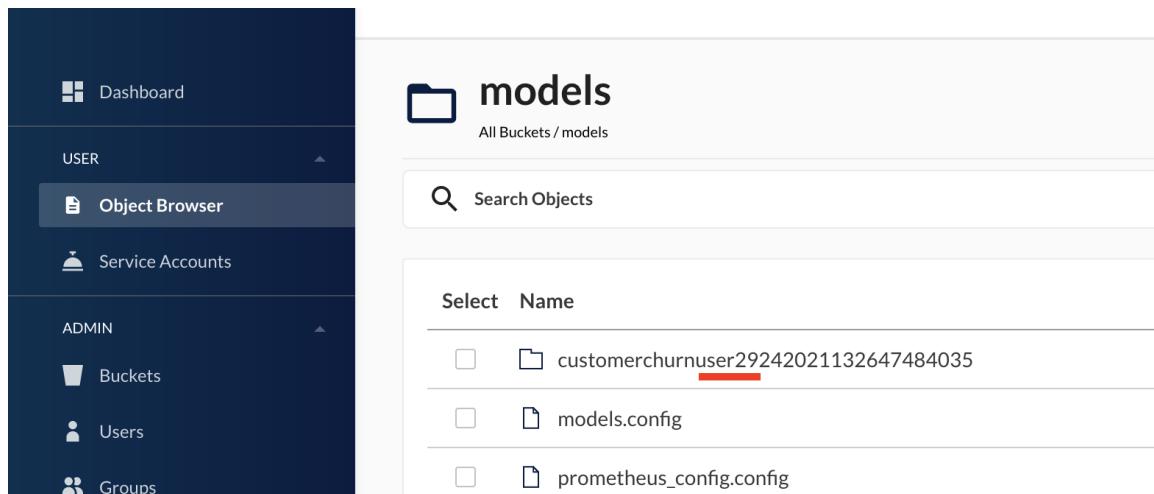
Open Minio, as described above in the section ***Instructions to access your prepared data file from the previous lab.***

Choose *Object Browser*, and open the *models* bucket:



The screenshot shows the Minio Console interface. On the left, the navigation sidebar includes sections for Dashboard, USER (with Object Browser selected and highlighted with a red box), Service Accounts, ADMIN (with Buckets, Users, Groups, IAM Policies, and Settings), and TOOLS. The main panel is titled "Object Browser" and displays the "All Buckets" list. A search bar at the top right says "Filter Buckets". Below it, a table lists buckets with their names: data, mlflow, model-stats, models (which is also highlighted with a red box), and rawdata.

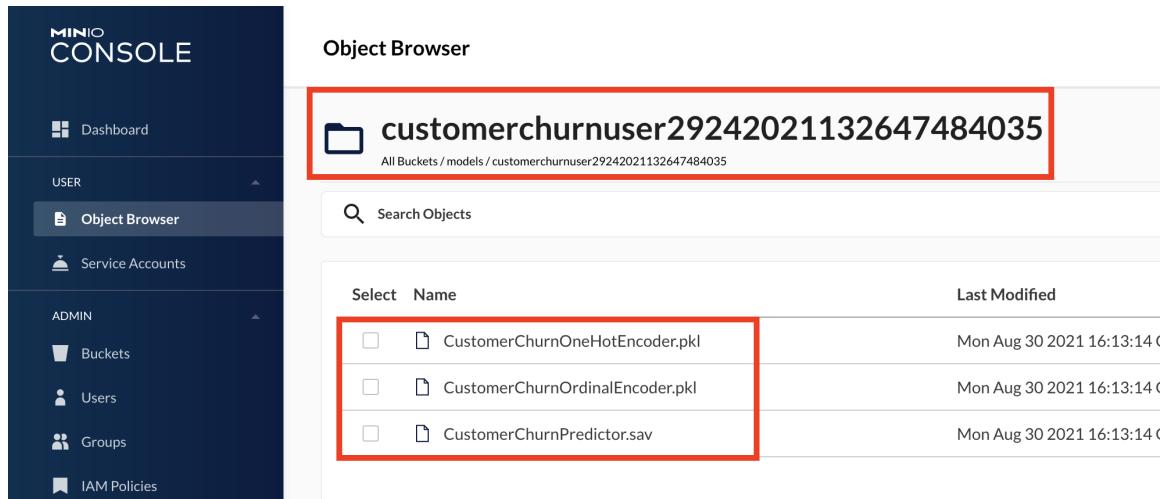
Find the folder containing your username – this is mine for user29. Click into yours.



The screenshot shows the Minio Object Browser with the "models" bucket selected. The left sidebar remains the same. The main panel now shows the contents of the "models" bucket under the heading "All Buckets / models". It includes a search bar labeled "Search Objects" and a table with columns "Select" and "Name". The table lists three items: "customerchurnuser29242021132647484035" (which is highlighted with a red box), "models.config", and "prometheus_config.config".

Note the binary artifacts that will be used to deploy your model to production in the next workshop ML OPs.

Also copy your experiment id, in my case `customerchurnuser29242021132647484035`. You need it in the ML OPs workshop.



The screenshot shows the MINIO Console Object Browser. On the left, the navigation sidebar includes 'Dashboard', 'USER' (with 'Object Browser' selected), 'Service Accounts', 'ADMIN' (with 'Buckets', 'Users', 'Groups', and 'IAM Policies' listed), and 'MINIO CONSOLE'. The main area is titled 'Object Browser' and shows a single bucket named 'customerchurnuser29242021132647484035'. Below the bucket name is a search bar labeled 'Search Objects'. A table lists three objects: 'CustomerChurnOneHotEncoder.pkl', 'CustomerChurnOrdinalEncoder.pkl', and 'CustomerChurnPredictor.sav'. All three objects are highlighted with a red box. The table has columns for 'Select', 'Name', and 'Last Modified'.

Select	Name	Last Modified
<input type="checkbox"/>	CustomerChurnOneHotEncoder.pkl	Mon Aug 30 2021 16:13:14 (0)
<input type="checkbox"/>	CustomerChurnOrdinalEncoder.pkl	Mon Aug 30 2021 16:13:14 (0)
<input type="checkbox"/>	CustomerChurnPredictor.sav	Mon Aug 30 2021 16:13:14 (0)