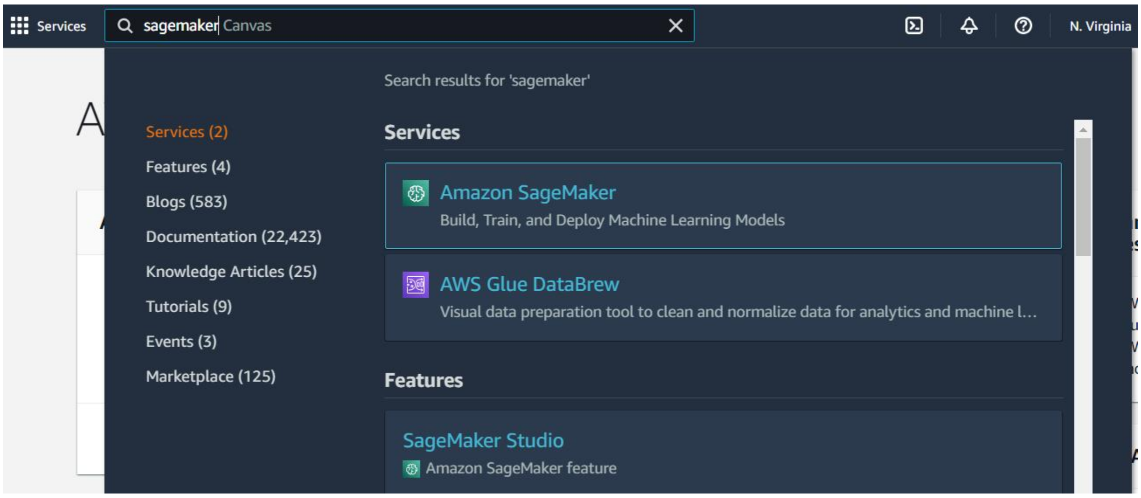
M12 Practical Challenge: Building a Supervised Learning Model via Amazon SageMaker Studio GUI

Table of Contents

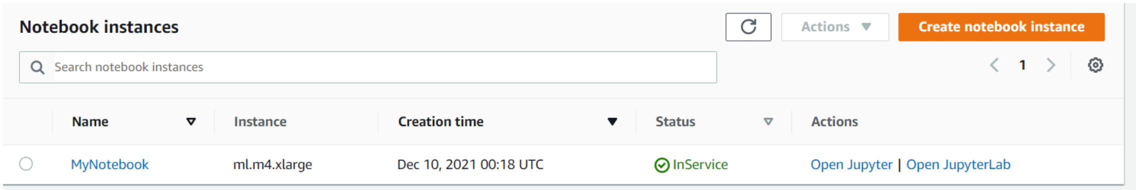
Alejandro C Parra Garcia

Amazon SageMaker

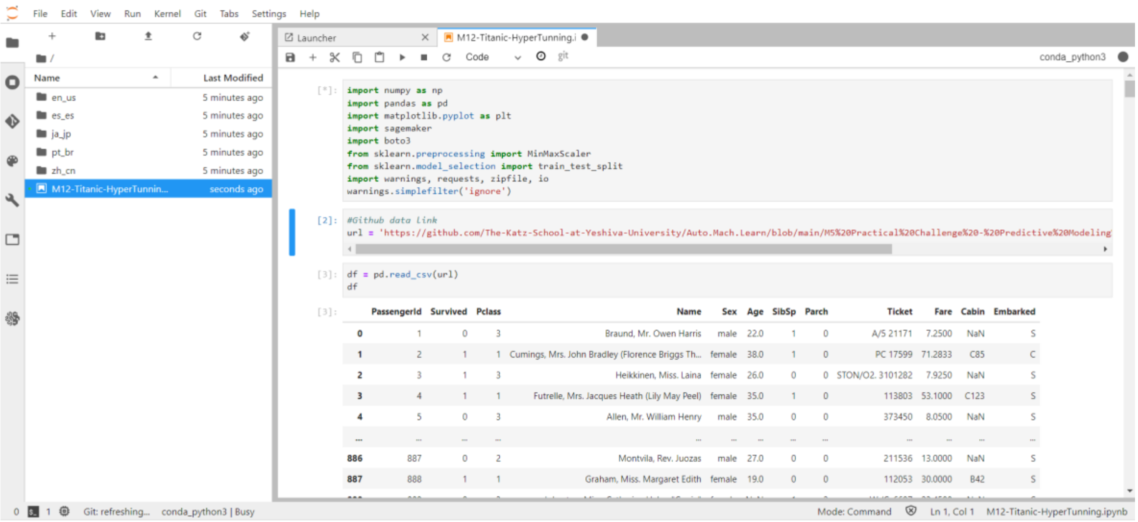
First thing we need to do is to open Amazon SageMaker and create a jupyter python notebook.



Open the Jupyter Lab



And now we can create the python notebook and add the code



Explore DataSet

I'm going to use the Titanic DataSet. This dataset was used in the "M5 Predictive Modeling in Python" part 2. So, most of the code is from there.

If I print it, it has the next appearance:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

We can also see the shape and the data types of the variables:

```
In [4]: df.shape
Out[4]: (891, 12)
```

```
In [5]: df.dtypes
Out[5]: PassengerId      int64
        Survived         int64
        Pclass           int64
        Name            object
        Sex             object
        Age            float64
        SibSp            int64
        Parch            int64
        Ticket          object
        Fare           float64
        Cabin           object
        Embarked        object
        dtype: object
```

There are 891 datapoints and 12 variables, including the target variable (Survived)

First thing lets check for nulls and treat them. There are nulls in 3 variables: Age, Cabin and Embarked. For age variable I'm going to impute it to the mean value. For the Cabin variable I'm going to change the nulls for 'NA'. Lastly, for the Embarked variable since there are only 2 nulls I'm going to delete them.

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
df.fillna(df.mean(), inplace=True)# Replace Age Nulls with Mean
df.loc[df["Cabin"].isnull(), 'Cabin'] = 'NA'# Replace Cabin Nulls with NA
df=df.dropna(axis=0,subset=['Embarked'])# Delete Embarked Nulls
```

Now let's treat the Numerical Data

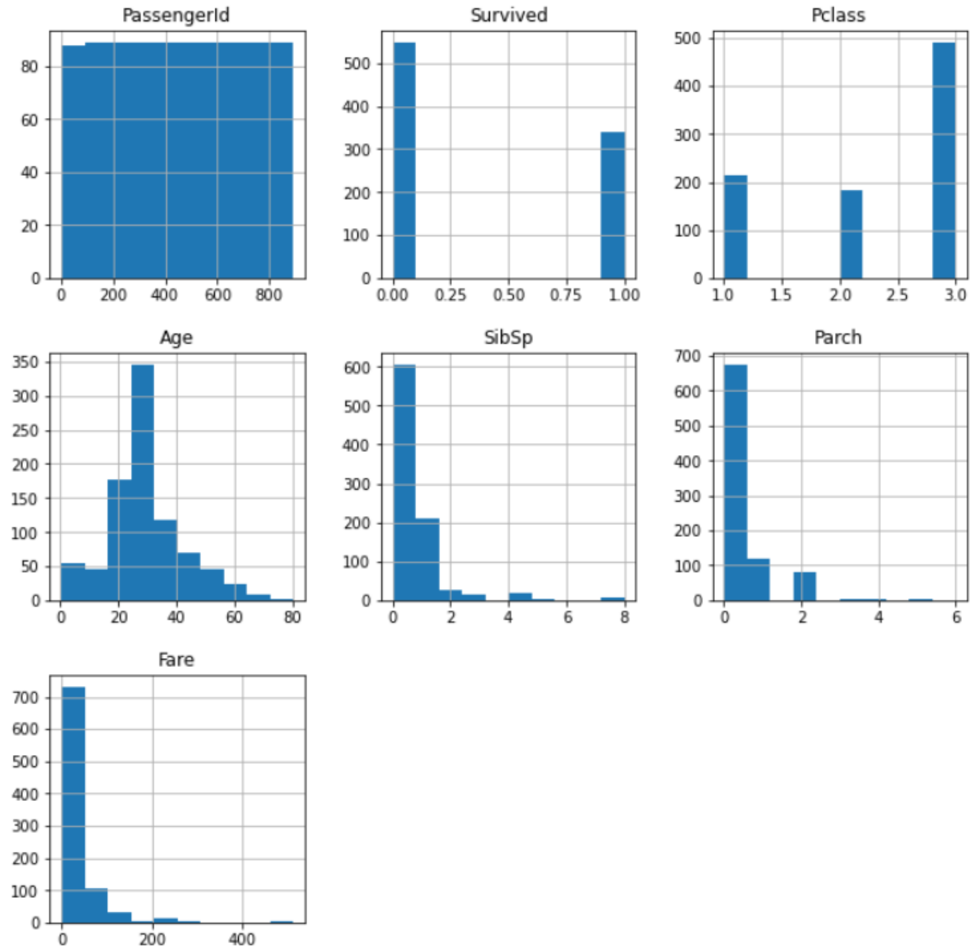
Treat Numerical Data

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000
mean	446.000000	0.382452	2.311586	29.653446	0.524184	0.382452	32.096681
std	256.998173	0.486260	0.834700	12.968366	1.103705	0.806761	49.697504
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	224.000000	0.000000	2.000000	22.000000	0.000000	0.000000	7.895800
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.000000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

We can plot them to see the distribution:

```
df[df.dtypes[(df.dtypes=="float64")|(df.dtypes=="int64")].index.values].hist(figsize=[11,11])
plt.show()
```



I'm going to drop PassangerId since it doesn't give any useful information, them I'm going to normalize:

```
#Eliminate the PassengerId column
df=df.drop(['PassengerId'], axis=1)
```

```
minmax=MinMaxScaler()
partB_minmax=minmax.fit_transform(df[['Pclass','Age','SibSp','Parch',
                                      'Fare']])
df_minmax = pd.DataFrame(partB_minmax)
df['Pclass']=df_minmax[0]
df['Age']=df_minmax[1]
df['SibSp']=df_minmax[2]
df['Parch']=df_minmax[3]
df['Fare']=df_minmax[4]
```

For the Categorical data, first we explore the data:

```
df[df.dtypes[(df.dtypes=="object")].index.values].columns
```

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

```
df[df.dtypes[(df.dtypes=="object")].index.values]
```

	Name	Sex	Ticket	Cabin	Embarked
0	Braund, Mr. Owen Harris	male	A/5 21171	NA	S
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C85	C
2	Heikkinen, Miss. Laina	female	STON/O2. 3101282	NA	S
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	Allen, Mr. William Henry	male	373450	NA	S
...	...	...	...	...	...
884	Montvila, Rev. Juozas	male	211536	NA	S
885	Graham, Miss. Margaret Edith	female	112053	B42	S
886	Johnston, Miss. Catherine Helen "Carrie"	female	W./C. 6607	NA	S
887	Behr, Mr. Karl Howell	male	111369	C148	C
888	Dooley, Mr. Patrick	male	370376	NA	Q

889 rows × 5 columns

Eliminate the Name and Ticket variables:

```
#Eliminate the Name column
df=df.drop(['Name'], axis=1)
#Eliminate the Ticket column
df=df.drop(['Ticket'], axis=1)
```

Modify the Cabin variable to delete the numbers of the cabins leaving only the Letter:

```
#Return the list of Letter, of the Cabins
def get_cabin_letter(cabin_string):
    splits=cabin_string.split(" ")
    res=set()
    for i in splits:
        if(i=="NA"):
            res.add(i)
        else:
            res.add(i[0])
    x=', '.join(res)
    return x
```

```
#Apply the changes
df['Cabin']= df.apply(lambda elem: get_cabin_letter(elem['Cabin']),axis=1)
```

Once that’s done we do a One hot Encoding with the rest of the variables:

```
#One hot Encoding on Sex, Cabin_Letter and Embarked
df=pd.get_dummies(df, columns=['Sex', 'Cabin', 'Embarked'], prefix=['Sex', 'Cabin', 'Embarked'] )
```

```
df
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Cabin_A	Cabin_B	...	Cabin_E	Cabin_F	Cabin_F, E	Cabin_F, G	Cabin_G	Ca
0	0	1.0	0.271174	0.125	0.000000	0.014151	0	1	0	0	...	0	0	0	0	0	
1	1	0.0	0.472229	0.125	0.000000	0.139136	1	0	0	0	...	0	0	0	0	0	
2	1	1.0	0.321438	0.000	0.000000	0.015469	1	0	0	0	...	0	0	0	0	0	
3	1	0.0	0.434531	0.125	0.000000	0.103644	1	0	0	0	...	0	0	0	0	0	
4	0	1.0	0.434531	0.000	0.000000	0.015713	0	1	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
884	0	0.5	0.334004	0.000	0.000000	0.025374	0	1	0	0	...	0	0	0	0	0	
885	1	0.0	0.233476	0.000	0.000000	0.058556	1	0	0	1	...	0	0	0	0	0	
886	0	1.0	0.367921	0.125	0.333333	0.045771	1	0	0	0	...	0	0	0	0	0	
887	1	0.0	0.321438	0.000	0.000000	0.058556	0	1	0	0	...	0	0	0	0	0	
888	0	1.0	0.396833	0.000	0.000000	0.015127	0	1	0	0	...	0	0	0	0	0	

889 rows × 22 columns

Splitting Data

Now we split the data into Training (80%), Validation (10%) and Test (10%) sets.

Splitting the data

```
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['Survived'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['Survived'])
```

```
print(train.shape)
print(test.shape)
print(validate.shape)
```

```
(711, 22)
(89, 22)
(89, 22)
```

```
print(train['Survived'].value_counts())
print(test['Survived'].value_counts())
print(validate['Survived'].value_counts())
```

```
0    439
1    272
Name: Survived, dtype: int64
0    55
1    34
Name: Survived, dtype: int64
0    55
1    34
Name: Survived, dtype: int64
```

Disclaimer:

Most of the code is from the Amazon Academy, Machine Learning Foundations module 3. It explains the functioning of Amazon Sage Maker. I adapted part of the code for the needs for this assignment. Since the works of the modules of Sagemaker, and the ways things are done, most code can be reutilise.

Upload to Amazon S3

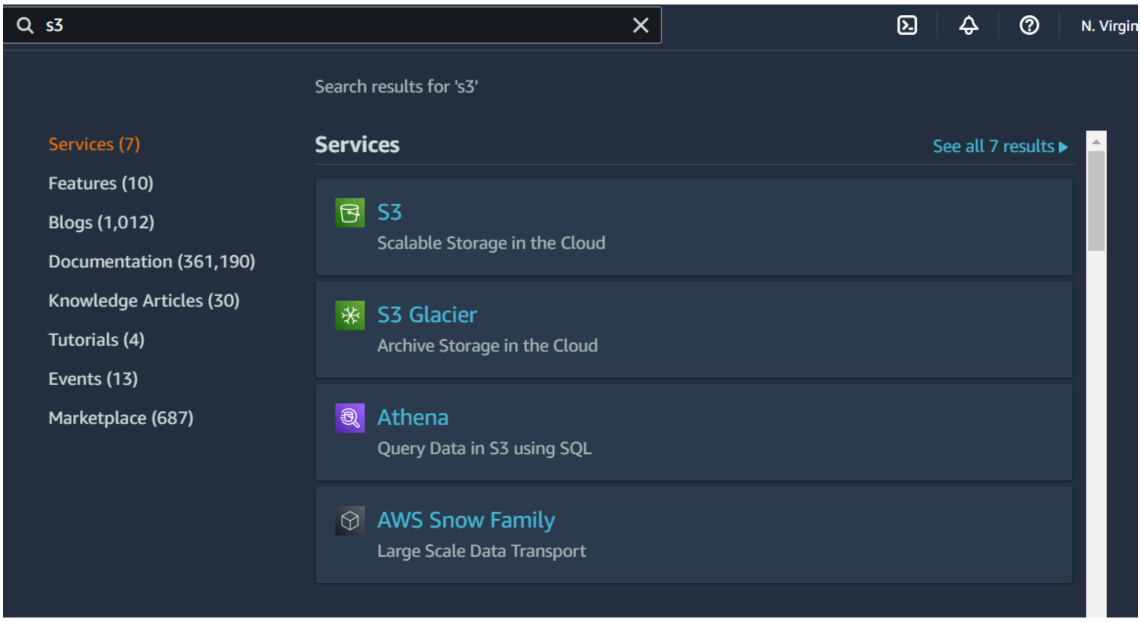
The 3 splits are uploaded to the Amazon S3, to the default Bucket, using the prefix: "demo-sagemaker-xgboost-titanic-prediction"

Uploading the data to Amazon S3

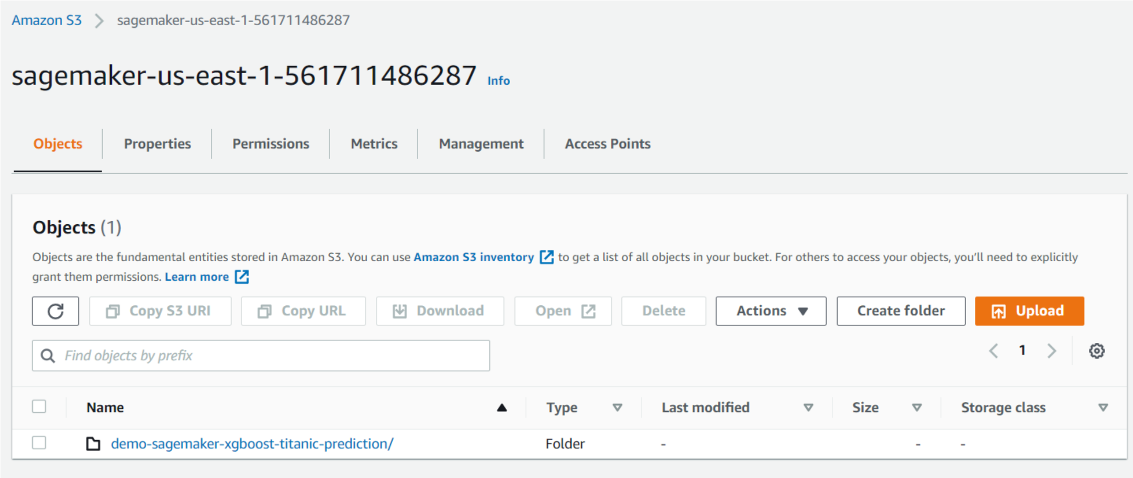
```python
bucket=sagemaker.Session().default_bucket()

prefix='demo-sagemaker-xgboost-titanic-prediction'

train_file='titanic_train.csv'
test_file='titanic_test.csv'
validate_file='titanic_validate.csv'

import os

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False)
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).put(Body=csv_buffer.getvalue())
```

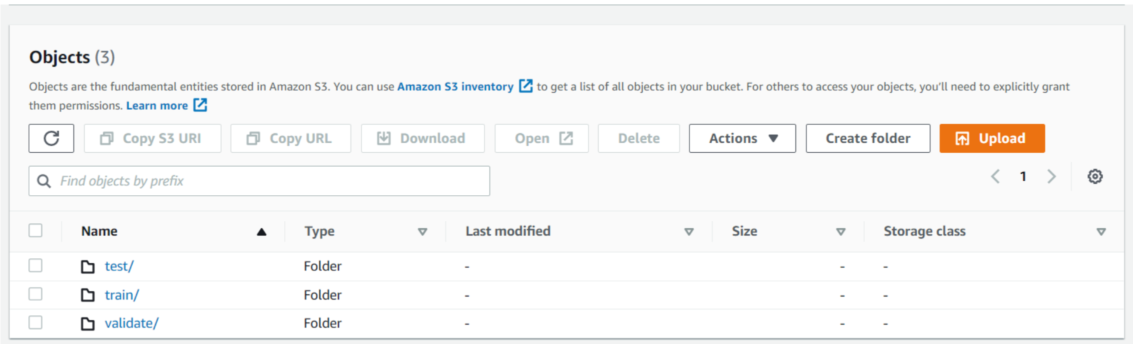
```python
upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

Now If I open the Amazon S3 module:



We can see that the file name "demo-sagemaker-xgboost-titanic-prediction" is there, if we open it we can see the 3 files of train, validate and test.


Amazon S3 > sagemaker-us-east-1-561711486287
sagemaker-us-east-1-561711486287 Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
demo-sagemaker-xgboost-titanic-prediction/
Folder
-
-
-


Objects (3)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
test/
Folder
-
-
-
train/
Folder
-
-
-
validate/
Folder
-
-
-

Train the Model

Now we need to train the model. Im going to use XBG model, for that I load it into a container.

Training the model

With HyperParameter Tunning Job

```
from sagemaker.image_uris import retrieve
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

Then we need to specify the output file, we create the model and set the Hyperparameters range for the SageMaker to tune and find the best model

```
# Output file
s3_output_location="s3://{}/{}/output/".format(bucket,prefix)

# Xgb Model
xgb=sagemaker.estimator.Estimator(container,
                                       sagemaker.get_execution_role(),
                                       instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       output_path=s3_output_location,
                                        sagemaker_session=sagemaker.Session())
#Hyperparameters objectives
xgb.set_hyperparameters(eval_metric='error@.40',
                        objective='binary:logistic',
                        num_round=42)
#Tunning Hyperparameters
hyperparameter_ranges = {'alpha': ContinuousParameter(0, 100),
                         'min_child_weight': ContinuousParameter(1, 5),
                         'subsample': ContinuousParameter(0.5, 1),
                         'eta': ContinuousParameter(0.1, 0.3),
                         'num_round': IntegerParameter(1,50)
                         }
#Metrics
objective_metric_name = 'validation:error'
objective_type = 'Minimize'
#Tuner
tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=5, # Set this to 10 or above depending upon budget & available time.
                            max_parallel_jobs=1,
                            objective_type=objective_type,
                            early_stopping_type='Auto')
```

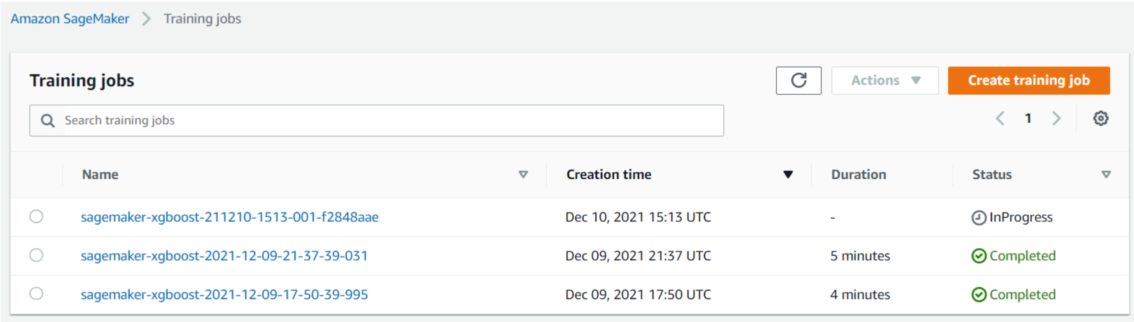
Once we have the model set, we specify the s3 input files and we train the model with the fit function.

```python
train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}
```

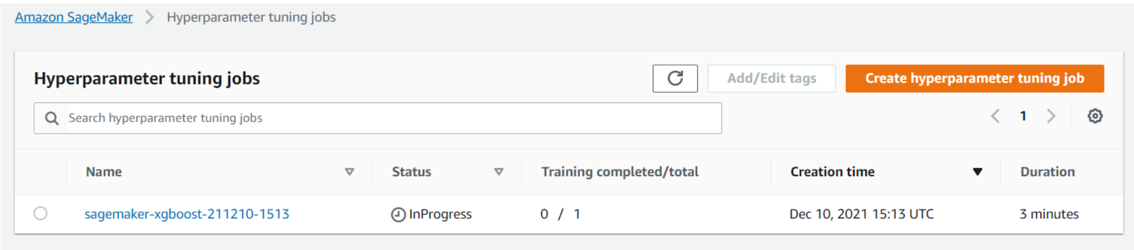
```python
tuner.fit(inputs=data_channels, logs=False)
tuner.wait()
```

Now we can check on the Amazon SageMaker the evolution of the training process.

If we check the training jobs, we can see the new job in progress



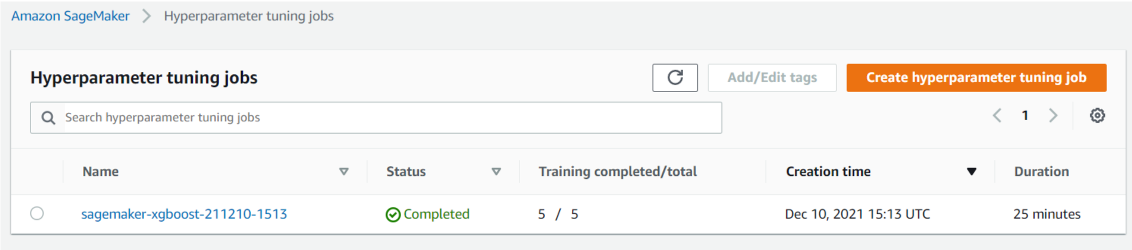
We can also check the hyperparameter tunning jobs tab. At this moment it has train 0/1 model. Since we specify 5 models to train, its going to keep running until those models are train.

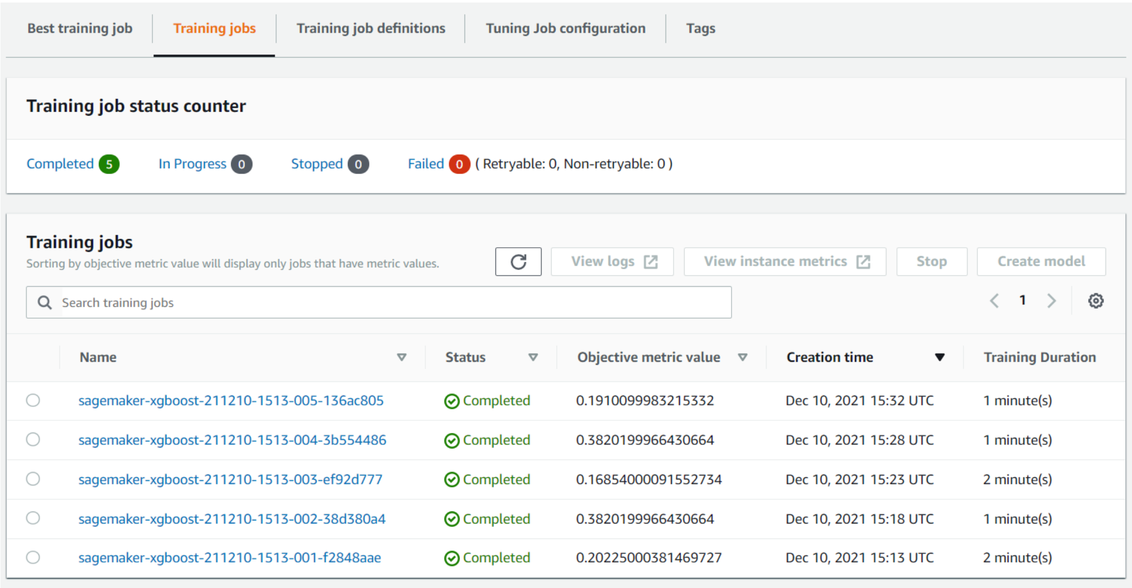


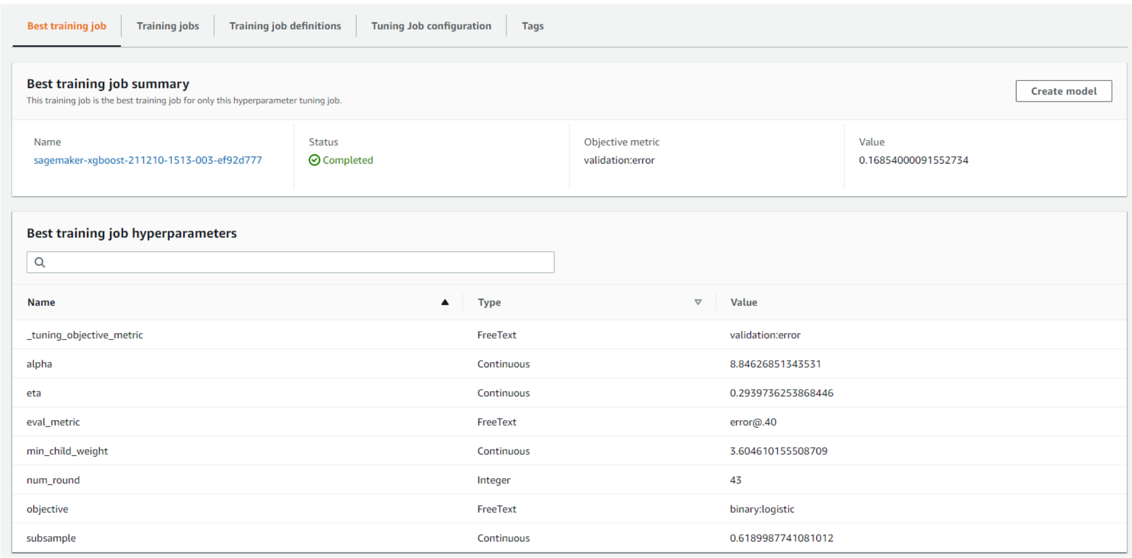
Now we need to wait for all the models to get train.

Best Model (Hyperparameter Tunning)

Once the process has finished, we can check for the best model. For that we can check them from the sagemaker interface or directly from code. If we check it from the Sagemaker interface we need to open the hyperparameter tunning and we can see al the jobs, we can also click on the best training job tab to see which model is the best. As we can see from the image:







We can also see it from code, we can retrieve all the jobs and print them:

Tuning job results

```
from pprint import pprint
from sagemaker.analytics import HyperparameterTuningJobAnalytics

tuner_analytics = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())

df_tuning_job_analytics = tuner_analytics.dataframe()

# Sort the tuning job analytics by the final metrics value
df_tuning_job_analytics.sort_values(
    by=['FinalObjectiveValue'],
    inplace=True,
    ascending=False if tuner.objective_type == "Maximize" else True)

# Show detailed analytics for the top 20 models
df_tuning_job_analytics.head(20)
```

```
attached_tuner = HyperparameterTuner.attach(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())
best_training_job = attached_tuner.best_training_job()
```

```
from sagemaker.estimator import Estimator
algo_estimator = Estimator.attach(best_training_job)

best_algo_model = algo_estimator.create_model(env={'SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT':"text/csv"})
```

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
2	8.846269	0.293974	3.604610	43.0	0.618999	sagemaker-xgboost-211210-1513-003-ef92d777	Completed	0.16854	2021-12-10 15:26:02+00:00	2021-12-10 15:27:48+00:00	106.0
0	1.592240	0.253176	2.773111	3.0	0.691381	sagemaker-xgboost-211210-1513-005-136ac805	Completed	0.19101	2021-12-10 15:36:35+00:00	2021-12-10 15:37:51+00:00	76.0
4	19.004594	0.286950	1.635211	31.0	0.882203	sagemaker-xgboost-211210-1513-001-f2848aae	Completed	0.20225	2021-12-10 15:16:15+00:00	2021-12-10 15:18:20+00:00	125.0
1	92.827592	0.293455	4.291666	2.0	0.501256	sagemaker-xgboost-211210-1513-004-3b554486	Completed	0.38202	2021-12-10 15:31:17+00:00	2021-12-10 15:32:31+00:00	74.0
3	80.099809	0.134340	2.123941	14.0	0.907649	sagemaker-xgboost-211210-1513-002-38d380a4	Completed	0.38202	2021-12-10 15:21:28+00:00	2021-12-10 15:22:40+00:00	72.0

Making Predictions

Once we have the best model, we load it and run the test dataset to make predictions. For that I'm using a bach transformation to predict for all the datapoints in the test dataset.

First, we need to delete the survived column, since it's the one we want to predict.

Performing a batch transform

```
batch_X = test.iloc[:,1:];
batch_X.head()
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Cabin_A	Cabin_B	Cabin_C	...	Cabin_E	Cabin_E, F	Cabin_F	Cabin_F, G	Cabin_G	Cabin_NA	Cabin_T
697	0.0	0.610455	0.125	0.166667	0.216430	0	1	0	0	1	...	0	0	0	0	0	0	0
633	1.0	0.107816	0.375	0.333333	0.054457	1	0	0	0	0	...	0	0	0	0	0	1	0
510	1.0	0.367921	0.000	0.000000	0.015713	0	1	0	0	0	...	0	0	0	0	0	1	0
866	1.0	0.367921	0.000	0.000000	0.018543	0	1	0	0	0	...	0	0	0	0	0	1	0
751	1.0	0.409399	0.000	0.000000	0.018543	0	1	0	0	0	...	0	0	0	0	0	1	0

5 rows × 21 columns

Then we upload the new data to the s3 bucket and run the model.

```
batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

```
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = best_algo_model.transformer(instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       strategy='MultiRecord',
                                       assemble_with='Line',
                                       output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                         data_type='S3Prefix',
                         content_type='text/csv',
                         split_type='Line')
xgb_transformer.wait()
```

If we check the batch transform jobs tab in sagemaker we can see the new one in progess.

Amazon SageMaker > Batch transform jobs

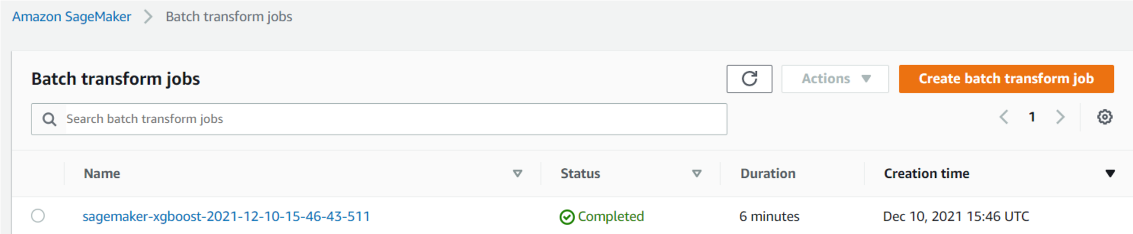
Batch transform jobs

Actions ▼ | Create batch transform job

Search batch transform jobs

Name	Status	Duration	Creation time
sagemaker-xgboost-2021-12-10-15-46-43-511	InProgress	2 minutes	Dec 10, 2021 15:46 UTC
sagemaker-xgboost-2021-12-09-21-56-04-953	Completed	6 minutes	Dec 09, 2021 21:56 UTC
sagemaker-xgboost-2021-12-09-17-55-07-647	Completed	6 minutes	Dec 09, 2021 17:55 UTC

Now that it has finished, we can check the results



For that we get the data from the s3 bucket

```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),',',names=['class'])
target_predicted.head(5)
```

	class
0	0.457696
1	0.392690
2	0.091962
3	0.091962
4	0.085662

We can see that the output is a probability distribution, so we can use a threshold to collapse it into a prediction.

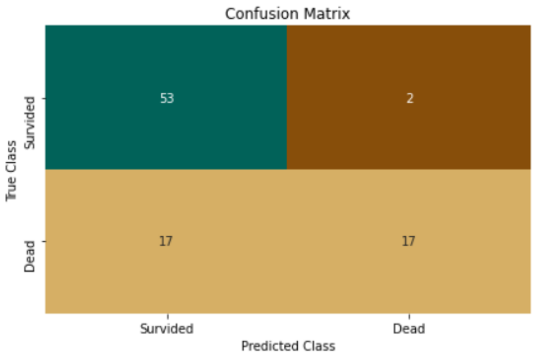
```
def binary_convert(x):
    threshold = 0.65
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['binary'] = target_predicted['class'].apply(binary_convert)

print(target_predicted.head(10))
test.head(10)
```

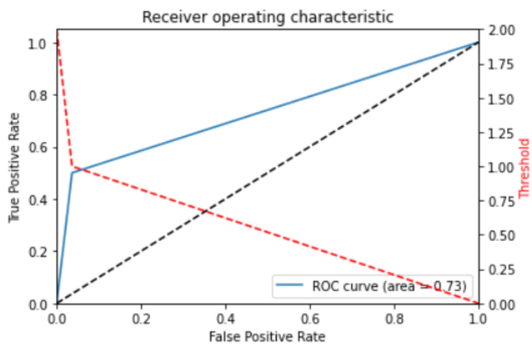
```
      class  binary
0  0.457696       0
1  0.392690       0
2  0.091962       0
3  0.091962       0
4  0.085662       0
5  0.428375       0
6  0.085700       0
7  0.460999       0
8  0.115513       0
9  0.083020       0
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Cabin_A	Cabin_B	...	Cabin_E	Cabin_E, F	Cabin_F	Cabin_F, G	Cabin_G	Cabin_NA	Cabin_T	Embarked_C	Embarked_Q	Embarked_S
697	0	0.0	0.610455	0.125	0.166667	0.216430	0	1	0	0	...	0	0	0	0	0	0	0	1	0	0
633	0	1.0	0.107816	0.375	0.333333	0.054457	1	0	0	0	...	0	0	0	0	0	1	0	0	0	1
510	0	1.0	0.367921	0.000	0.000000	0.015713	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1
866	0	1.0	0.367921	0.000	0.000000	0.018543	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1
751	0	1.0	0.409399	0.000	0.000000	0.018543	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1
61	0	0.0	0.560191	0.125	0.000000	0.162932	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
400	0	1.0	0.321438	0.000	0.000000	0.015713	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1
350	0	0.0	0.367921	0.000	0.000000	0.068315	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
862	0	0.5	0.296306	0.000	0.000000	0.025374	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1
159	0	1.0	0.547625	0.000	0.166667	0.031425	0	1	0	0	...	0	0	0	0	0	1	0	0	0	1

Confusion Matrix and ROC

We now can plot the Confusion Matrix to see the results





We can see that the model failed to classify 19 out of the 89 datapoints. That's an accuracy of 78.65%. We can also see that the AUC of the ROC is 0.73. The model is a decent model, but in needs improvements. In order to perform better. The model misclassified half of the non-survival into survived class. Which indicates that the model is good at predicting if someone survived but not good to predict if someone was dead or not.