

# Banking Subscription Project

(Complete end-to-end project by Ravi Chaurasia)

(Mail me at [ravichaurasia700@gmail.com](mailto:ravichaurasia700@gmail.com))

Follow GitHub link for complete code

<https://github.com/ravichaurasia/Banking-Subscription>

## Table of Contents

1	Overview.....	3
1.1	Reference Paper.....	3
1.2	Data Set Information.....	3
1.3	Problem Statement:.....	3
1.4	Attribute Information:.....	3
2	Proposed Methodology.....	4
2.1	Technical Specification .....	4
2.1.1	Physical.....	4
2.1.2	Software.....	4
2.2	Feature Summary.....	4
2.3	Data Cleaning .....	5
2.4	Feature Engineering.....	5
2.4.1	Feature Encoding.....	5
2.5	Exploratory Data Analysis .....	6
2.6	Model Run .....	6
2.7	Results.....	7
2.8	Coding Details.....	7
2.9	Platforms/Tools Used (if any) .....	7
3	Prerequisites.....	8
4	Project Building.....	8
4.1	Installation .....	8
4.2	Project folder information.....	8
4.3	Project Architecture .....	9
4.4	I/O and Schema Files information .....	9
4.5	Training Data Validation.....	11
4.6	main.py file (entry point of code).....	12
4.7	Training Data Insertion in Database .....	14
4.8	Model Training.....	14
4.9	Prediction Data Description.....	14
4.10	Prediction Data Validation .....	15
4.11	Prediction Data Insertion in Database .....	15
4.12	Prediction.....	15
5	Banking subscription project folder structure .....	16
6	Deployment.....	16
6.1	Deployment to Azure:.....	17
6.2	Command Prompt steps .....	22

# 1 Overview

## 1.1 Reference Paper

Citation Request: This dataset is public available for research. The details are described in [Moro et al., 2014]. Please include this citation if you plan to use this database:

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

[pdf] <http://dx.doi.org/10.1016/j.dss.2014.03.001>

## 1.2 Data Set Information

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Dataset link: - <http://archive.ics.uci.edu/ml/datasets/Bank+Marketing>

## 1.3 Problem Statement:

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

## 1.4 Attribute Information:

**Input variables:**

**# bank client data:**

**1 - age** (numeric)

**2 - job**: type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')

**3 - marital**: marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)

**4 - education** (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')

**5 - default**: has credit in default? (categorical: 'no', 'yes', 'unknown')

**6 - housing**: has housing loan? (categorical: 'no', 'yes', 'unknown')

**7 - loan**: has personal loan? (categorical: 'no', 'yes', 'unknown')

**# related with the last contact of the current campaign:**

**8 - contact**: contact communication type (categorical: 'cellular', 'telephone')

**9 - month**: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

**10 - day\_of\_week**: last contact day of the week (categorical: 'mon', 'tue', 'wed', 'thu', 'fri')

**11 - duration**: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

**# other attributes:**

**12 - campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)

**13 - pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

**14 - previous:** number of contacts performed before this campaign and for this client (numeric)

**15 - poutcome:** outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

# social and economic context attributes

**16 - emp.var.rate:** employment variation rate - quarterly indicator (numeric)

**17 - cons.price.idx:** consumer price index - monthly indicator (numeric)

**18 - cons.conf.idx:** consumer confidence index - monthly indicator (numeric)

**19 - euribor3m:** euribor 3 month rate - daily indicator (numeric)

**20 - nr.employed:** number of employees - quarterly indicator (numeric)

**Output variable (desired target):**

**21 - y** - has the client subscribed a term deposit? (binary: 'yes','no')

## 2 Proposed Methodology

### 2.1 Technical Specification

#### 2.1.1 Physical

S.No.	Machine Type	Model	Processor	RAM	GPU
1.	Local	Lenovo Ideapad 320	I3 (2 Cores)	12 Gb	None
2.	Server	Google Colab			Yes

#### 2.1.2 Software

S.No.	OS/ Software	Version	Details (any specific)	URL
1.	Windows 10			
2.				

### 2.2 Feature Summary

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age                41188 non-null int64
job                41188 non-null object
marital            41188 non-null object
education          41188 non-null object
default            41188 non-null object
housing            41188 non-null object
loan               41188 non-null object
contact            41188 non-null object
month              41188 non-null object
day_of_week        41188 non-null object
duration           41188 non-null int64
campaign           41188 non-null int64
pdays             41188 non-null int64
previous           41188 non-null int64
poutcome           41188 non-null object
emp.var.rate       41188 non-null float64
cons.price.idx     41188 non-null float64
cons.conf.idx      41188 non-null float64
euribor3m          41188 non-null float64
nr.employed        41188 non-null float64
y                  41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

## 2.3 Data Cleaning

S.No.	Column Name	Treatment	Details
1.	None	Missing Value	
2.			

## 2.4 Feature Engineering

### 2.4.1 Feature Encoding

S.No.	Column Name	Encoding Method	Details
1.	Job, Marital, Education, Default, Housing, Loan,	Response Encoding	<a href="https://medium.com/@thewingedwolf.winterfell/response-coding-for-categorical-data-7bb8916c6dc1">https://medium.com/@thewingedwolf.winterfell/response-coding-for-categorical-data-7bb8916c6dc1</a>

	Contact, Month, Poutcome		
2.			

## 2.5 Exploratory Data Analysis

- Suggest Hypothesis
- Assess assumptions
- Univariate analysis for both numerical and categorical columns
- Correlations, Skewness, Kurtosis etc.
- Imbalanced dataset handle by upsampling
- Support for model selection

Detailed analysis can be seen at notebook file

## 2.6 Model Run

Run No.	Model	Metric	Value	Hyperparameter values
1.	XGB Classifier	Accuracy	Train:73.8% Test: 75%	base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=100, n_jobs=1, nthread=None, objective='binary:logistic', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1
2.	LGBM Classifier	Accuracy	Train: 77.79% Test: 76.92%	boosting_type='gbdt', class_weight=None, colsample_bytree=1.0, importance_type='split', learning_rate=0.1, max_depth=-1, min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0, n_estimators=100, n_jobs=-1, num_leaves=31, objective=None, random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=1.0, subsample_for_bin=200000, subsample_freq=0

3.	CatBoost Classifier	Accuracy	Train: 97.84% Test: 93.38%	iterations=2000, learning_rate=0.1, depth=8, eval_metric='Accuracy', random_seed = 0, bagging_temperature = 0.2, od_type='Iter', metric_period = 75, od_wait=100
4.	Stacking (LGB, CatBoost and SVM)	Accuracy	Test: 92.98%	

## 2.7 Results

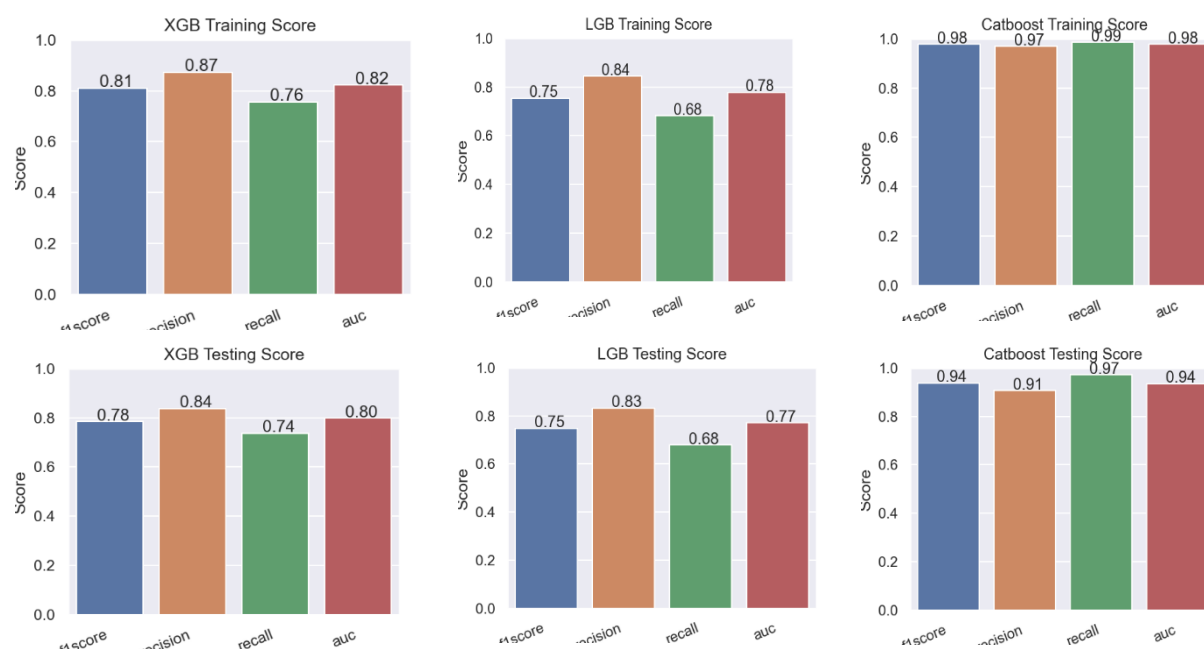


Figure 1: Training and testing score

## 2.8 Coding Details

S.No.	Programming Language	Package Used	Details
1.	Python	Catboost, Lgboost, XGboost	Pip install <name>
2.	Python	Scikit learn	GridSearchCV, model_selection, metrics

## 2.9 Platforms/Tools Used (if any)

S.No.	Platform Tool	Details
1.	H2O driveless AI	

### 3 Prerequisites

- 1) Anaconda installed
- 2) Python Basic
- 3) Working knowledge of any IDE (Pycharm, Spyder and etc.)
- 4) Basic flask knowledge
- 5) Postman installed

## 4 Project Building

### 4.1 Installation

- `conda create -n BankingSubscription python==3.6.8`
- `conda active BankingSubscription`

\* Located to project folder

- `pip install -r requirements`

### 4.2 Project folder information

- 1) Here, two code are built; one without using clustering approach and other with clustering approach. Other than both are same (`Code` and `Code_with Clustring` folder).
- 2) `Notebook/ML3BankingSubscription_EDA_V2.ipynb`  
It consists of complete EDA performed for project.
- 3) `Notebook/tst_encode.json` file contains the response encoding value.
- 4) As well as reference paper and problem statement document is also provided for more information.



## 4.3 Project Architecture

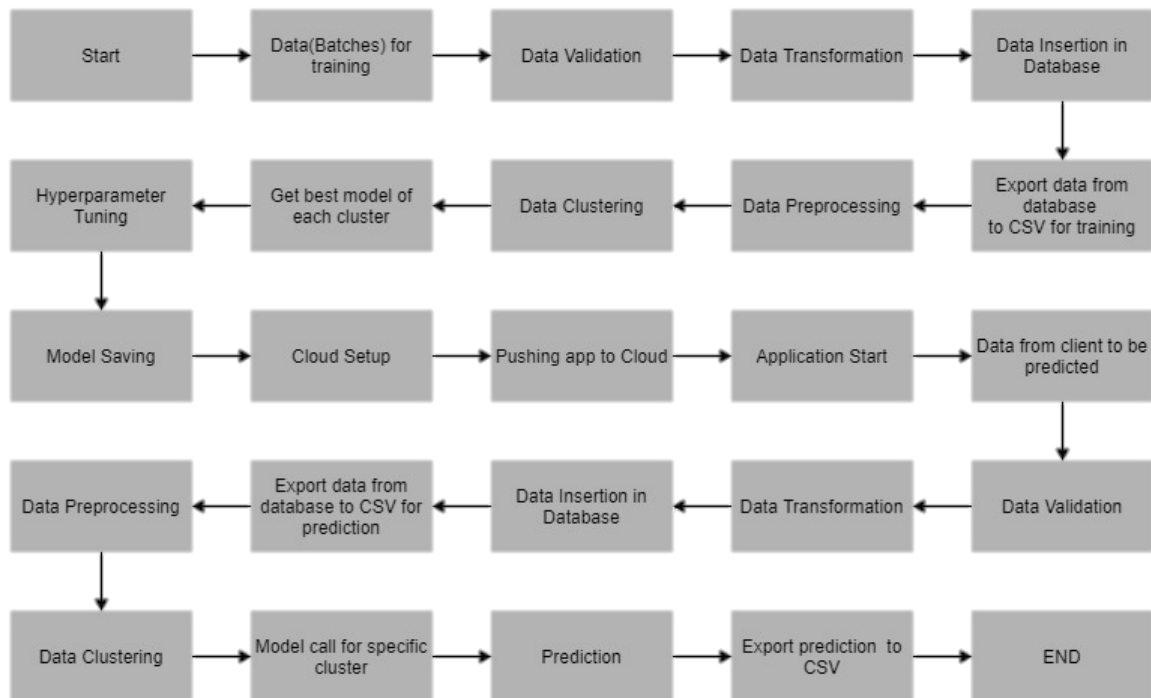


Figure 2: Complete project architecture

## 4.4 I/O and Schema Files information

- Input for training can be putted in any folder but I have putted in **RawData** folder with name **bank-additional-full.csv** (File name should be same)

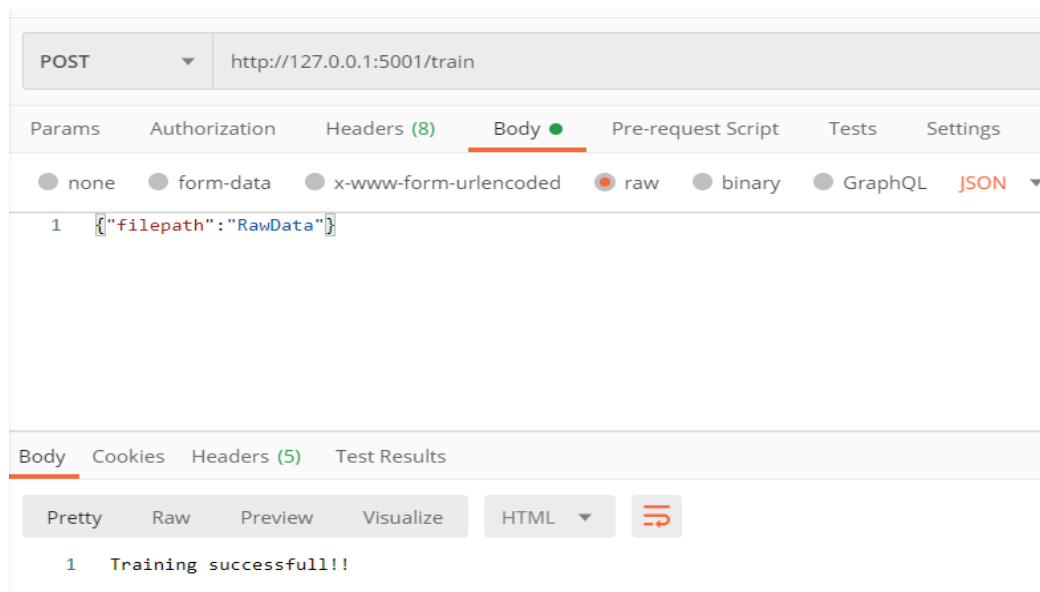


Figure 3: Training input file hit from postman server

- Input for prediction can be putted in any folder but I have putted in **Prediction\_Data** folder with name **test.csv** (File name should be same)
- Output of prediction is generated in **Prediction\_Output\_File** with name **Predictions.csv**

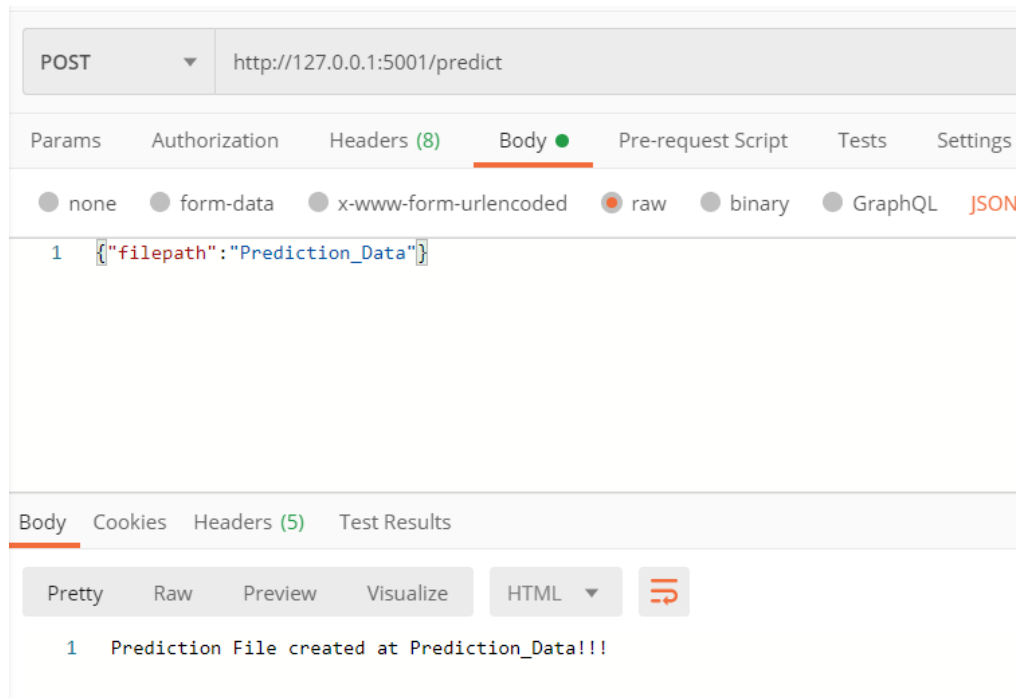


Figure 4 Prediction input file hit from postman server

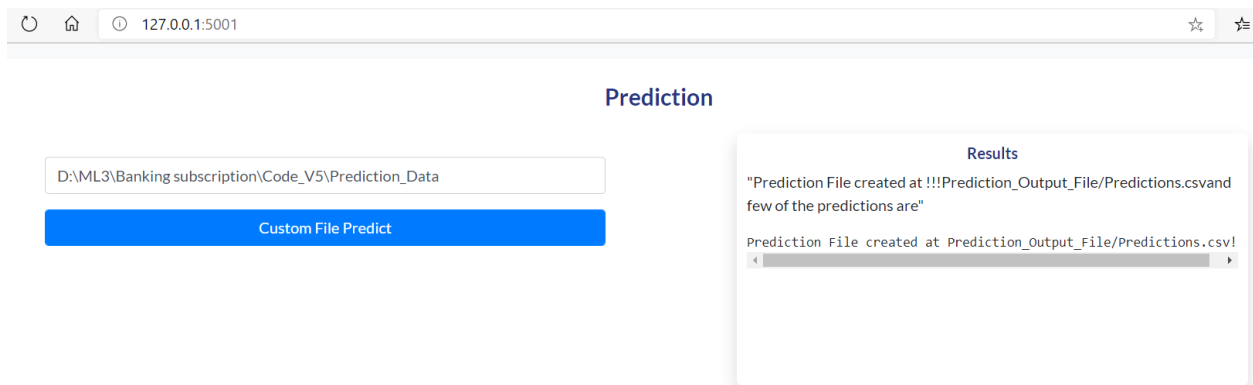


Figure 5: Prediction input file hit from local web server

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as: Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns, and their datatype. Shown in figure 5: Training schema file (a) and Prediction schema file (b)

Input of training and prediction should match with these schema file, then only it will process continue.

```

1  schema_training.json x
2  {
3    "SampleFileName": "bank-additional-full.csv",
4    "LengthOfDateStampInFile": 0,
5    "LengthOfTimeStampInFile": 0,
6    "NumberOfColumns": 21,
7    "ColName":
8    {
9      "age": "int64",
10     "job": "object",
11     "marital": "object",
12     "education": "object",
13     "default": "object",
14     "housing": "object",
15     "loan": "object",
16     "contact": "object",
17     "month": "object",
18     "day_of_week": "object",
19     "duration": "int64",
20     "campaign": "int64",
21     "pdays": "int64",
22     "previous": "int64",
23     "poutcome": "object",
24     "emp.var.rate": "float64",
25     "cons.price.idx": "float64",
26     "cons.conf.idx": "float64",
27     "euribor3m": "float64",
28     "nr.employed": "float64",
29     "y": "object"
30   }
31 }
32

```

(a)

```

1  schema_prediction.json x
2  {
3    "SampleFileName": "test.csv",
4    "LengthOfDateStampInFile": 0,
5    "LengthOfTimeStampInFile": 0,
6    "NumberOfColumns": 20,
7    "ColName":
8    {
9      "age": "int64",
10     "job": "object",
11     "marital": "object",
12     "education": "object",
13     "default": "object",
14     "housing": "object",
15     "loan": "object",
16     "contact": "object",
17     "month": "object",
18     "day_of_week": "object",
19     "duration": "int64",
20     "campaign": "int64",
21     "pdays": "int64",
22     "previous": "int64",
23     "poutcome": "object",
24     "emp.var.rate": "float64",
25     "cons.price.idx": "float64",
26     "cons.conf.idx": "float64",
27     "euribor3m": "float64",
28     "nr.employed": "float64"
29   }
30 }
31

```

(b)

Figure 6: Training schema file (a) and Prediction schema file (b)

## 4.5 Training Data Validation

In this step, we perform different sets of validation on the given set of training files.

1. **Name Validation**- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good\_Data\_Folder" else we move such files to "Bad\_Data\_Folder."
2. **Number of Columns** - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad\_Data\_Folder."
3. **Name of Columns** - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad\_Data\_Folder".

4. **The datatype of columns** - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Bad\_Data\_Folder".
5. **Null values in columns** - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Bad\_Data\_Folder".

## 4.6 main.py file (entry point of code)

```
main.py x
1  from flask import Flask, request, render_template
2  from flask import Response
3  from flask_cors import CORS, cross_origin
4  import flask_monitoringdashboard as dashboard
5  import os
6  from prediction_Validation_Insertion import pred_validation
7  from predictFromModel import prediction
8  from trainingModel import trainModel
9  from training_Validation_Insertion import train_validation
10
11  os.putenv('LANG', 'en_US.UTF-8')
12  os.putenv('LC_ALL', 'en_US.UTF-8')
13
14  app = Flask(__name__)
15  dashboard.bind(app)
16  CORS(app)
17
18  @app.route("/", methods=['GET']) # Main Page
19  @cross_origin() # To handle various route
20  def home():
21      return render_template('index.html')
22
23  @app.route("/predict", methods=['POST']) # Prediction Route
24  @cross_origin()
25  def predictRouteClient():
26      try:
27          if request.json is not None: # prection file access from Postman server
28
29              path = request.json['filepath'] # Obtained file path
30
31              pred_val = pred_validation(path) # object initialization
32
```

```

main.py x
32
33     pred_val.prediction_validation()_# calling the prediction_validation function
34
35     pred = prediction(path)_# object initialization
36
37     path = pred.predictionFromModel()_# predicting for dataset using saved model
38
39     return Response("Prediction File created at %s!!!" % path)_# Predicted file path
40
41 elif request.form is not None:_# prection file access from Web API
42
43     path = request.form['filepath']_# Obtained file path
44
45     pred_val = pred_validation(path)_# object initialization
46
47     pred_val.prediction_validation()_# calling the prediction_validation function
48
49     pred = prediction(path)_# object initialization
50
51     path = pred.predictionFromModel()_# predicting for dataset using saved model
52
53     return Response("Prediction File created at %s!!!" % path)_# Predicted file path
54
55 except ValueError:
56     return Response("Error Occurred! %s" %ValueError)
57 except KeyError:
58     return Response("Error Occurred! %s" %KeyError)
59 except Exception as e:
60     return Response("Error Occurred! %s" %e)
61
62 @app.route("/train", methods=['POST'])_# Training Route
63 @cross_origin()

main.py x
63 @cross_origin()
64 def trainRouteClient():
65     try:
66         if request.json['filepath'] is not None:_# prection file access from Postman server (Main folder file:- RawData)
67
68         path = request.json['filepath']_# Obtained file path
69
70         train_valObj = train_validation(path)_# object initialization
71
72         train_valObj.train_validation()_# calling the training_validation function
73
74         trainModelObj = trainModel()_# object initialization
75
76         trainModelObj.trainingModel()_# training the model for the files in the table
77
78     except ValueError:
79         return Response("Error Occurred! %s" % ValueError)
80     except KeyError:
81         return Response("Error Occurred! %s" % KeyError)
82     except Exception as e:
83         return Response("Error Occurred! %s" % e)
84     return Response("Training successfull!!")
85
86 port = int(os.getenv("PORT",5001))_# port obtained for local machine
87
88 if __name__ == "__main__":
89     # app.run(debug=True)_# Enable when upload to production
90     app.run(port=port,debug=True)_# Disable when upload to production
91

```

Figure 7: main.py file screenshot, explanation of lines are given right side respectively

## 4.7 Training Data Insertion in Database

- 1) **Database Creation and connection** - Create a database with the given name passed. If the database is already created, open the connection to the database.
- 2) **Table creation in the database** - Table with name - "Good\_Data", is created in the database for inserting the files in the "Good\_Data\_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.
- 3) **Insertion of files in the table** - All the files in the "Good\_Data\_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad\_Data\_Folder".

## 4.8 Model Training

- 1) **Data Export from Db** - The data in a stored database is exported as a CSV file to be used for model training.
- 2) **Data Preprocessing**
  - a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.
  - b) Replace the invalid values with numpy "nan" so we can use imputer on such values.
  - c) Encode the categorical values
  - d) Check for null values in the columns. If present, impute the null values using the Single imputer.
- 3) **Clustering** - KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using "KneeLocator" function. The idea behind clustering is to implement different algorithms  
  
To train data in different clusters. The Kmeans model is trained over preprocessed data and the model is saved for further use in prediction.
- 4) **Model Selection** - After clusters are created, we find the best model for each cluster. We are using three algorithms, "LGBost", "XGBoost" and "CatBoost". For each cluster, all three algorithms are passed with the best parameters derived from GridSearch. We calculate the AUC scores for all models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

## 4.9 Prediction Data Description

Prediction file with name "test.csv" will contain 20 columns of feature information.

Apart from prediction files, we also require a "schema" file which contains all the relevant information about the training files such as:

Name of the files, Number of Columns, Name of the Columns and their datatype.

## 4.10 Prediction Data Validation

In this step, we perform different sets of validation on the given set of training files.

- 1) **Name Validation**- We validate the name of the files on the basis of given Name in the schema file. We have created a regex pattern as per the name given in schema file, to use for validation. After validating the pattern in the name, we check for length of date in the file name as well as length of time in the file name. If all the values are as per requirement, we move such files to "Good\_Data\_Folder" else we move such files to "Bad\_Data\_Folder".
- 2) **Number of Columns** - We validate the number of columns present in the files, if it doesn't match with the value given in the schema file then the file is moved to "Bad\_Data\_Folder".
- 3) **Name of Columns** - The name of the columns is validated and should be same as given in the schema file. If not, then the file is moved to "Bad\_Data\_Folder".
- 4) **Datatype of columns** - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If datatype is wrong then the file is moved to "Bad\_Data\_Folder".
- 5) **Null values in columns** - If any of the columns in a file has all the values as NULL or missing, we discard such file and move it to "Bad\_Data\_Folder".

## 4.11 Prediction Data Insertion in Database

- 1) **Database Creation and connection** - Create database with the given name passed. If the database is already created, open the connection to the database.
- 2) **Table creation in the database** - Table with name - "Good\_Data", is created in the database for inserting the files in the "Good\_Data\_Folder" on the basis of given column names and datatype in the schema file. If table is already present then new table is not created, and new files are inserted the already present table as we want training to be done on new as well old training files.
- 3) **Insertion of files in the table** - All the files in the "Good\_Data\_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad\_Data\_Folder".

## 4.12 Prediction

- 1) **Data Export from Db** - The data in the stored database is exported as a CSV file to be used for prediction.
- 2) **Data Preprocessing**
  - a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.
  - b) Replace the invalid values with numpy "nan" so we can use imputer on such values.
  - c) Encode the categorical values
  - d) Check for null values in the columns. If present, impute the null values using the Single imputer.

3) **Clustering** - KMeans model created during training is loaded, and clusters for the preprocessed prediction data is predicted.

4) **Prediction** - Based on the cluster number, the respective model is loaded and is used to predict the data for that cluster.

Once the prediction is made for all the clusters, the predictions along with the original names before label encoder are saved in a CSV file at a given location and the location is returned to the screen.

## 5 Banking subscription project folder structure

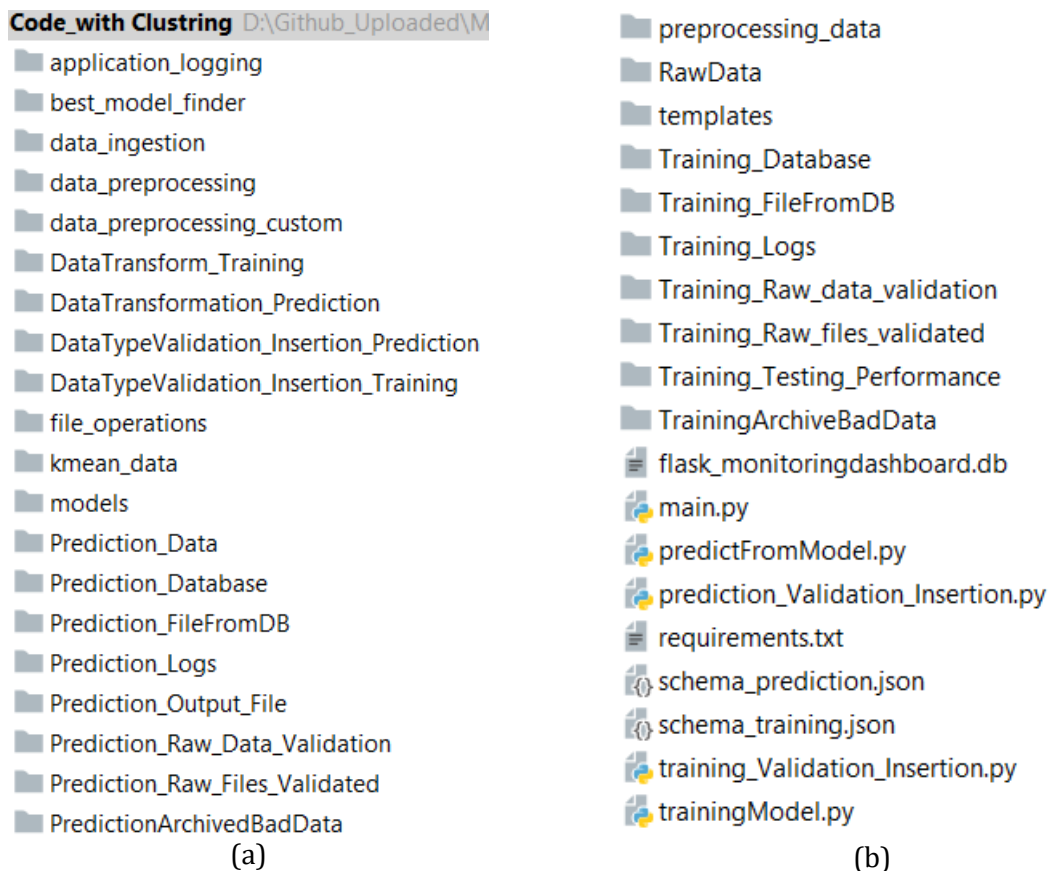


Figure 8: Folder structure of code

## 6 Deployment

- We will be deploying the model to the Azure Web Services Platform.
- **requirements.txt** file consists of all the packages that you need to deploy the app in the cloud.



## 6.1 Deployment to Azure:

- Go to <https://portal.azure.com/> and create an account if already haven't created one.
- Go to the Azure account and create a web app.

1 Identity verification by card

2 Agreement

☐ I agree to the [subscription agreement](#), [offer details](#), and [privacy statement](#).

I will receive information, tips, and offers about Azure, including Azure Newsletter, Pricing updates, and other Microsoft products and services.

Sign up

Confirming your information...

What's included

✓

**12 months of free products**  
Get free access to popular products like *virtual machines*, *storage*, and *databases* in your first 30 days, and for 12 months after you upgrade your account to pay-as-you-go pricing.

✓

**₹13,300 credit**  
Use your ₹13,300 credit to experiment with any Azure service in your first 30 days—beyond the free product amounts.

✓

**25+ always-free products**  
Take advantage of more than 25 products, including *serverless*, *containers*, and *artificial intelligence*, that are always free. Get these in your first 30 days, and always—once you choose to upgrade.

✓

**No automatic charges**  
You won't be charged unless you choose to upgrade. Before the end of your first 30 days, you'll be notified and have the chance to upgrade and start paying only for the resources you use beyond the free amounts.

Figure 9 Free account include details (debit visa card can be used to create account if not have any credit card)

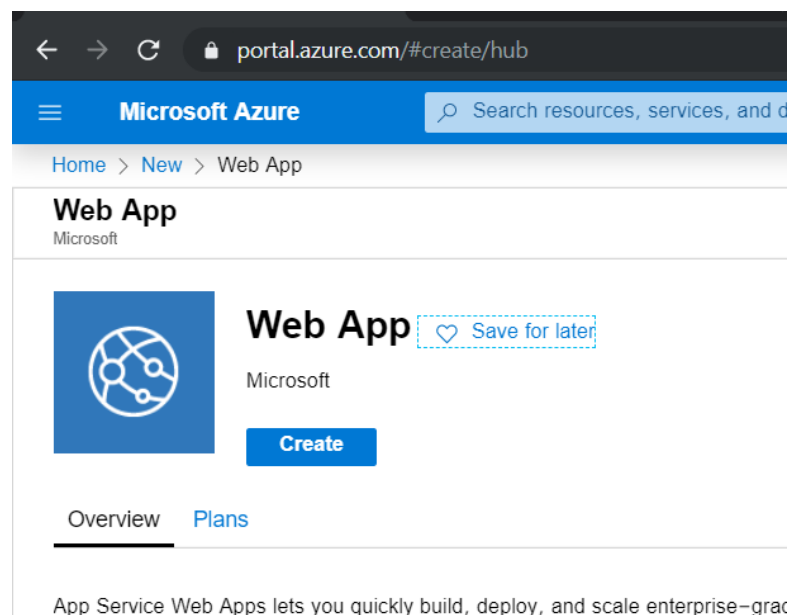


Figure 10 select create web app

- Provide the app name, resource group (create new if necessary), runtime stack(Python 3.7), region, select the 1 GB size, which is free to use. Click **Review+create** to create the web app.

# Web App

Basics   Monitoring   Tags   Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

## Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ

Resource Group \* ⓘ

[Create new](#)

## Instance Details

Name \*  .azurewebsites.net

Publish \* ☒ Code ☐ Docker Container

Runtime stack \*

Operating System \* ☒ Linux ☐ Windows

Region \*

[Review + create](#)

[< Previous](#)

[Next : Monitoring >](#)

Figure 11: Creating Web App

Microsoft Azure

Search resources, services, and docs (G+/)

Home > Web App

Instance Details

Name \*  .azurewebsites.net

Publish \* ☒ Code ☐ Docker Container

Runtime stack \*

Operating System \* ☒ Linux ☐ Windows

Region \*

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Linux Plan (Central US) \* ⓘ

[Create new](#)

Sku and size \* 

Free F1  
1 GB memory  
[Change size](#)

Spec Picker

Dev / Test  
For less demanding workloads

Production  
For most production workloads

The first Basic (B1) core for Linux is free for the first 30 days!

Recommended pricing tiers

<b>F1</b> 1 GB memory 60 minutes/day compute Free	<b>B1</b> 100 total ACU 1.75 GB memory A-Series compute equivalent 868.50 INR/Month (Estimated)
--	---

[See additional options](#)

Included hardware

Every instance of your App Service plan will configuration:

- Memory**  
Memory available to run application App Service plan.
- Storage**  
1 GB disk storage shared by all app plan.

[Review + create](#) [< Previous](#) [Next : Monitoring >](#) [Apply](#)

Figure 12: Select 1 GB memory option for free usage


Microsoft Azure

Search resources, services, and docs (G+)

[Home >](#)

# Web App

### Summary

**Web App**  
by Microsoft

### Details

Subscription	
Resource Group	
Name	
Publish	Code
Runtime stack	Python 3.6

### App Service Plan (New)

Name	
Operating System	Linux
Region	Central US
SKU	Free
ACU	Shared infrastructure
Memory	1 GB memory

### Monitoring

Application Insights	Not enabled
----------------------	-------------

Create

< Previous

Next >

[Download a template for automation](#)

Figure 13: Click create to process further

Microsoft Azure

Search resources, services, and docs (G+)

🔍 📄 🗨️ ⚙️ ? 😊

[Home >](#)

## Microsoft.Web-WebApp-Portal-963b4f53-9b57 | Overview

Deployment

🔍 Search (Ctrl+/)

«

🗑️ Delete

🛑 Cancel

🔄 Redeploy

🔄 Refresh


🌿 Overview

👤 Inputs

📄 Outputs

📄 Template

... Your deployment is underway

 Deployment name: Microsoft.Web-WebApp-Portal-963b4f53-9b57

Start time: 6/12/2020, 1:51:19 AM

Subscription: [Free Trial](#)

Correlation ID: d239cc1c-ef05-429a-9356-1131728ee999

Resource group:

[Deployment details \(Download\)](#)

Resource	Type	Status	Operation details
No results.			

 Security Center

Secure your resources

[Go to Azure Security Center](#)

Free Microsoft Start learning

Figure 14: Wait for some time till deployment has done

- Once the deployment is completed, open the app and go to the 'Deployment Center' option. Select 'local git' for source control and click continue.

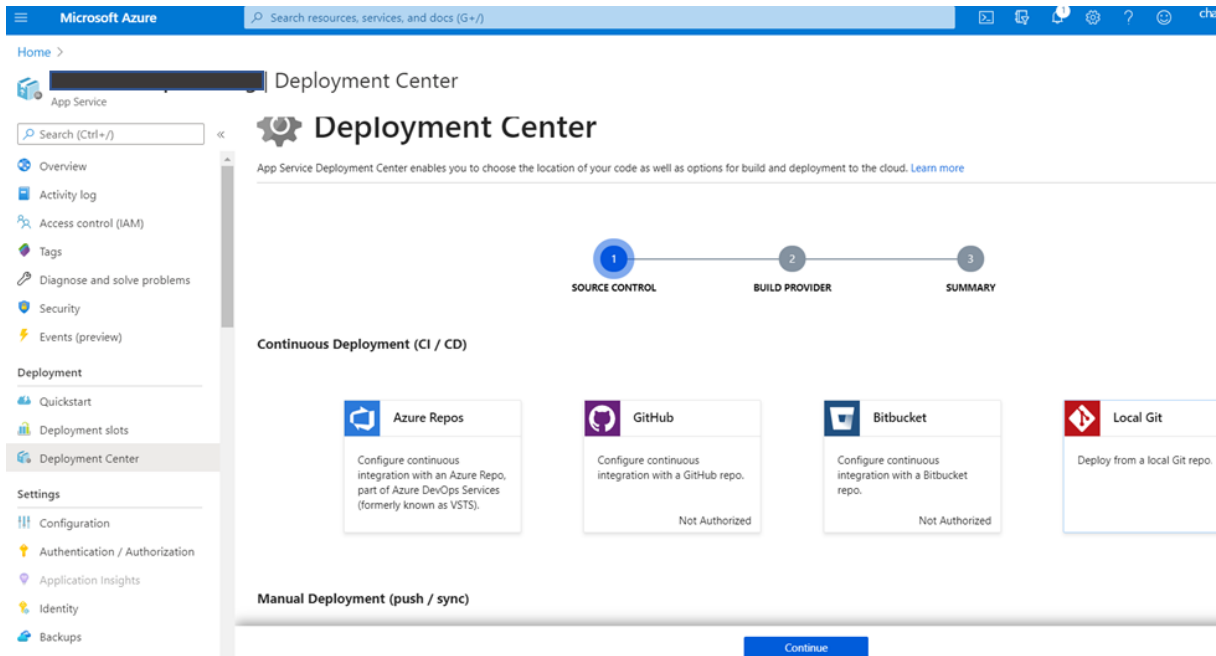


Figure 15: Select Local Git option to push from local environment

- Once Local Git option is selected, then select the kudo 'App service build provider' as the build provider and click continue.

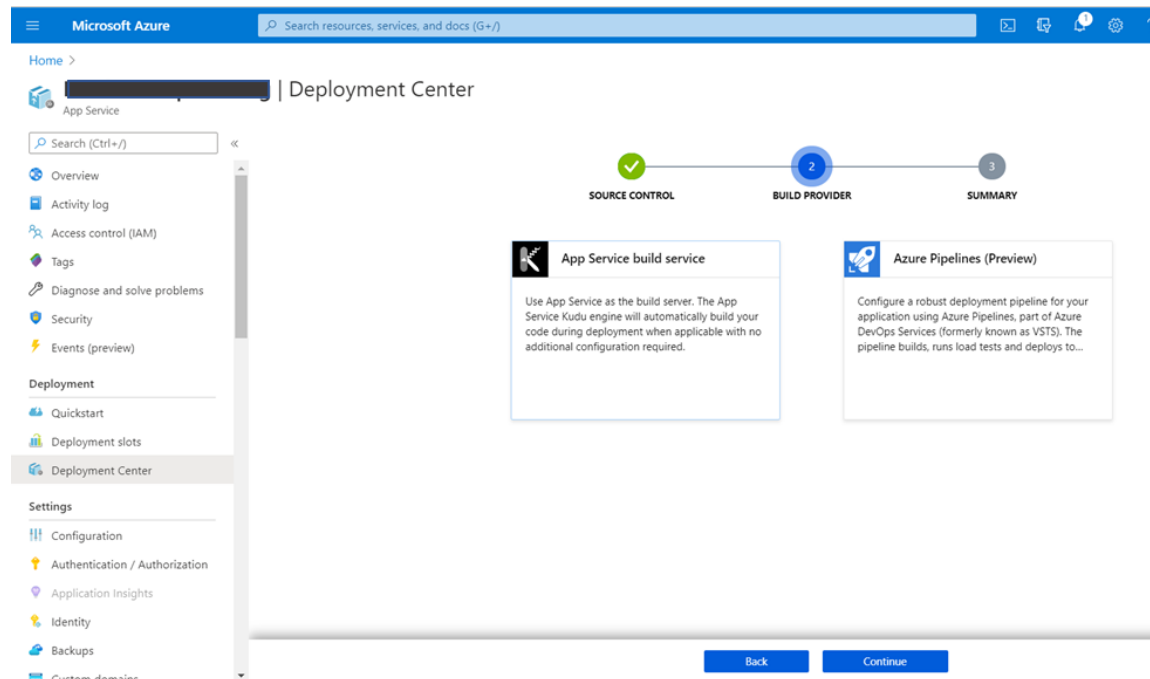


Figure 16: Select kudo App service build and continue

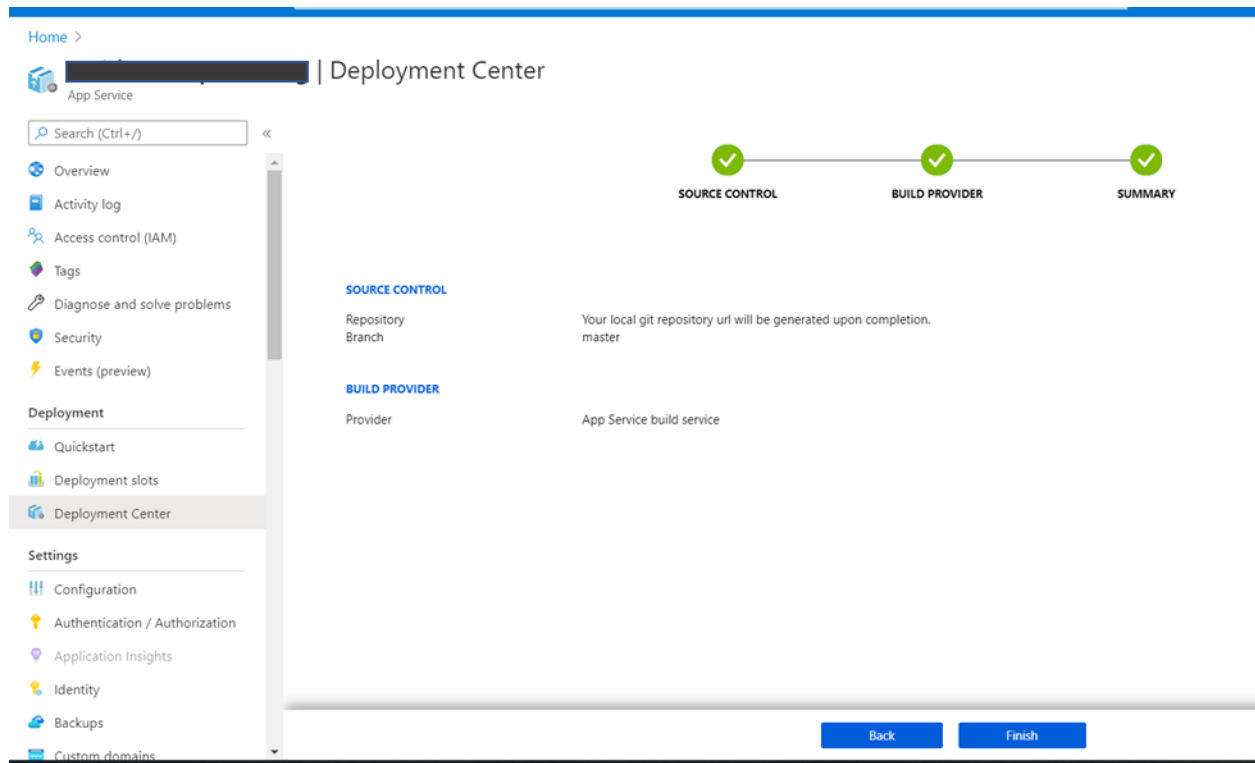


Figure 17: Click finish to complete deployment center process

- Click 'Finish' to complete the setup.

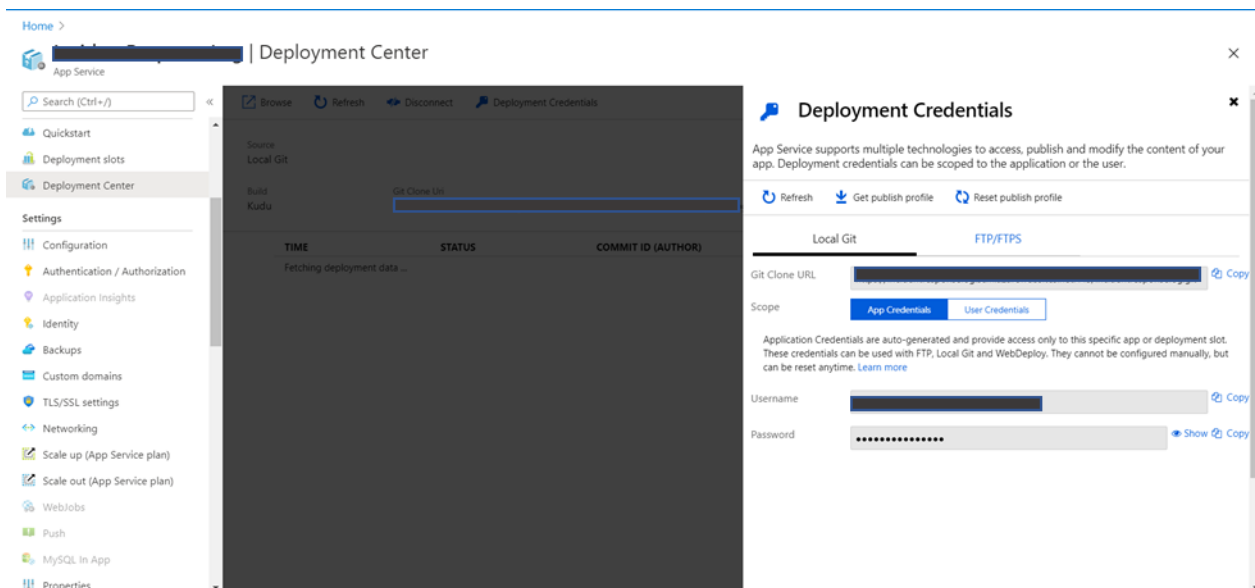


Figure 18: Copy the URL, username and password

- Go to the overview section of the app, and the Git link now will be visible.
- Go to 'Deployment Credentials' and copy the username and password. These will be required when doing the final push to the remote git repository.

## 6.2 Command Prompt steps

- Open a command prompt and navigate to your project folder.
- Run `git init` to initialise an empty git repository
- Create a new remote git alias using the command: `git remote add <alias> <git clone url>`
- Use `git add .` to add all the files to the local git repository.
- Use `git commit -m "First Commit"` to commit the code to the git repo.
- Push the code to the remote repo using `git push <alias> master -f`
- This prompts for a username and password. Provide the same credentials as copied in the step above.

After deployment, from the 'overview' section, copy the URL and paste into the browser to see the application running.