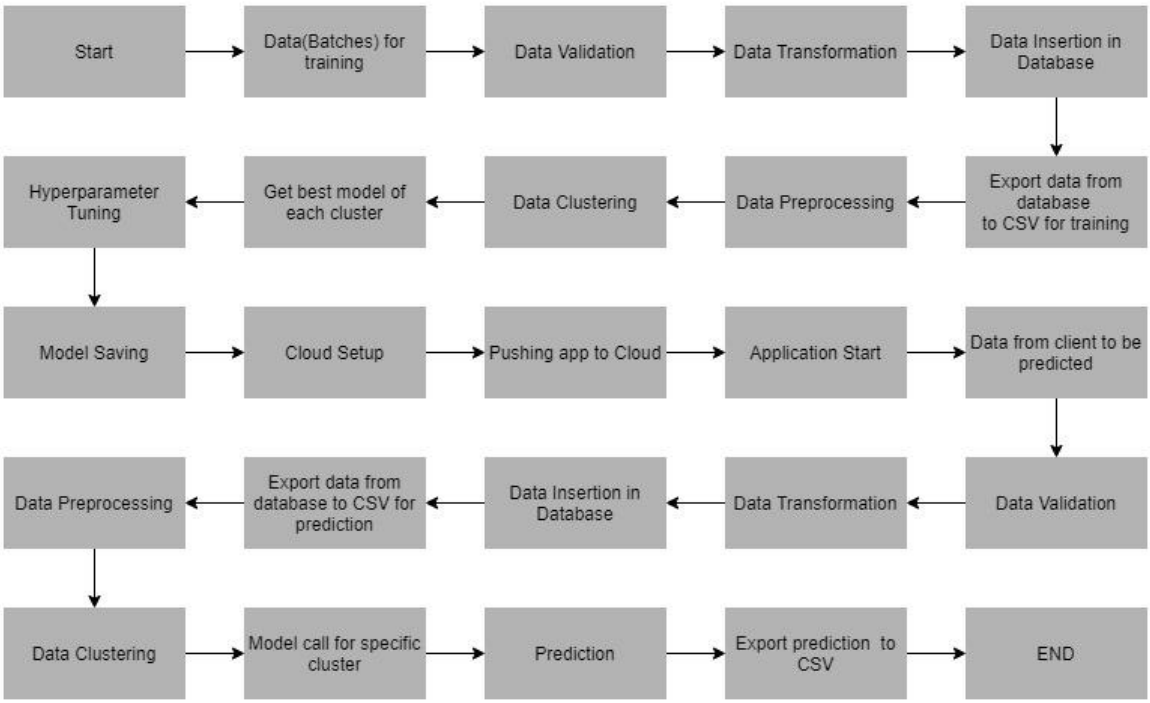
Problem Statement

To build a classification methodology to determine whether a customer is placing a fraudulent insurance claim.

Architecture



Data Description

The client will send data in multiple sets of files in batches at a given location. The data has been extracted from the census bureau.

The data contains the following attributes:

Features:

1. **months_as_customer:** It denotes the number of months for which the customer is associated with the insurance company.
2. **age:** continuous. It denotes the age of the person.
3. **policy_number:** The policy number.
4. **policy_bind_date:** Start date of the policy.
5. **policy_state:** The state where the policy is registered.

6. **policy_csl**-combined single limits. How much of the bodily injury will be covered from the total damage. https://www.berkshireinsuranceservices.com/arecombinedsinglelimitsbetter
7. **policy_deductable**: The amount paid out of pocket by the policy-holder before an insurance provider will pay any expenses.
8. **policy_annual_premium**: The yearly premium for the policy.
9. **umbrella_limit**: An umbrella insurance policy is extra liability insurance coverage that goes beyond the limits of the insured's homeowners, auto or watercraft insurance. It provides an additional layer of security to those who are at risk of being sued for damages to other people's property or injuries caused to others in an accident.
10. **insured_zip**: The zip code where the policy is registered.
11. **insured_sex**: It denotes the person's gender.
12. **insured_education_level**: The highest educational qualification of the policy-holder.
13. **insured_occupation**: The occupation of the policy-holder.
14. **insured_hobbies**: The hobbies of the policy-holder.
15. **insured_relationship**: Dependents on the policy-holder.
16. **capital-gain**: It denotes the monitory gains by the person.
17. **capital-loss**: It denotes the monitory loss by the person.
18. **incident_date**: The date when the incident happened.
19. **incident_type**: The type of the incident.
20. **collision_type**: The type of collision that took place.
21. **incident_severity**: The severity of the incident.
22. **authorities_contacted**: Which authority was contacted.
23. **incident_state**: The state in which the incident took place.
24. **incident_city**: The city in which the incident took place.
25. **incident_location**: The street in which the incident took place.

26. **incident_hour_of_the_day:** The time of the day when the incident took place.
27. **property_damage:** If any property damage was done.
28. **bodily_injuries:** Number of bodily injuries.
29. **Witnesses:** Number of witnesses present.
30. **police_report_available:** Is the police report available.
31. **total_claim_amount:** Total amount claimed by the customer.
32. **injury_claim:** Amount claimed for injury
33. **property_claim:** Amount claimed for property damage.
34. **vehicle_claim:** Amount claimed for vehicle damage.
35. **auto_make:** The manufacturer of the vehicle
36. **auto_model:** The model of the vehicle.
37. **auto_year:** The year of manufacture of the vehicle.

Target Label:

Whether the claim is fraudulent or not.

38. fraud_reported: Y or N

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.

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. It 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".

Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database has already been created, open a 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".

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 the columns not required for prediction.
b) For this dataset, the null values were replaced with '?' in the client data. Those '?' have been replaced with NaN values.
c) Check for null values in the columns. If present, impute the null values using the categorical imputer.
d) Replace and encode the categorical values with numeric values.
e) Scale the numeric values using the standard scaler.

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

The Kmeans model is trained over preprocessed data, and the model is saved for further use in prediction.

4) **Model Selection –** After the clusters have been created, we find the best model for each cluster. We are using two algorithms, "SVM" and "XGBoost". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the AUC scores for both 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.

Prediction Data Description

The Client will send the data in multiple sets of files in batches at a given location. Data will contain the annual income of various persons.

Apart from prediction 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.

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 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 the timestamp 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 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 the datatype is incorrect, 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".

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 a new table is not created, and new files are inserted into 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".

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 the columns not required for prediction.
b) For this dataset, the null values were replaced with '?' in the client data. Those '?' have been replaced with NaN values.
c) Check for null values in the columns. If present, impute the null values using the categorical imputer.
d) Replace and encode the categorical values with numeric values.
e) Scale the numeric values using the standard scaler.

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.

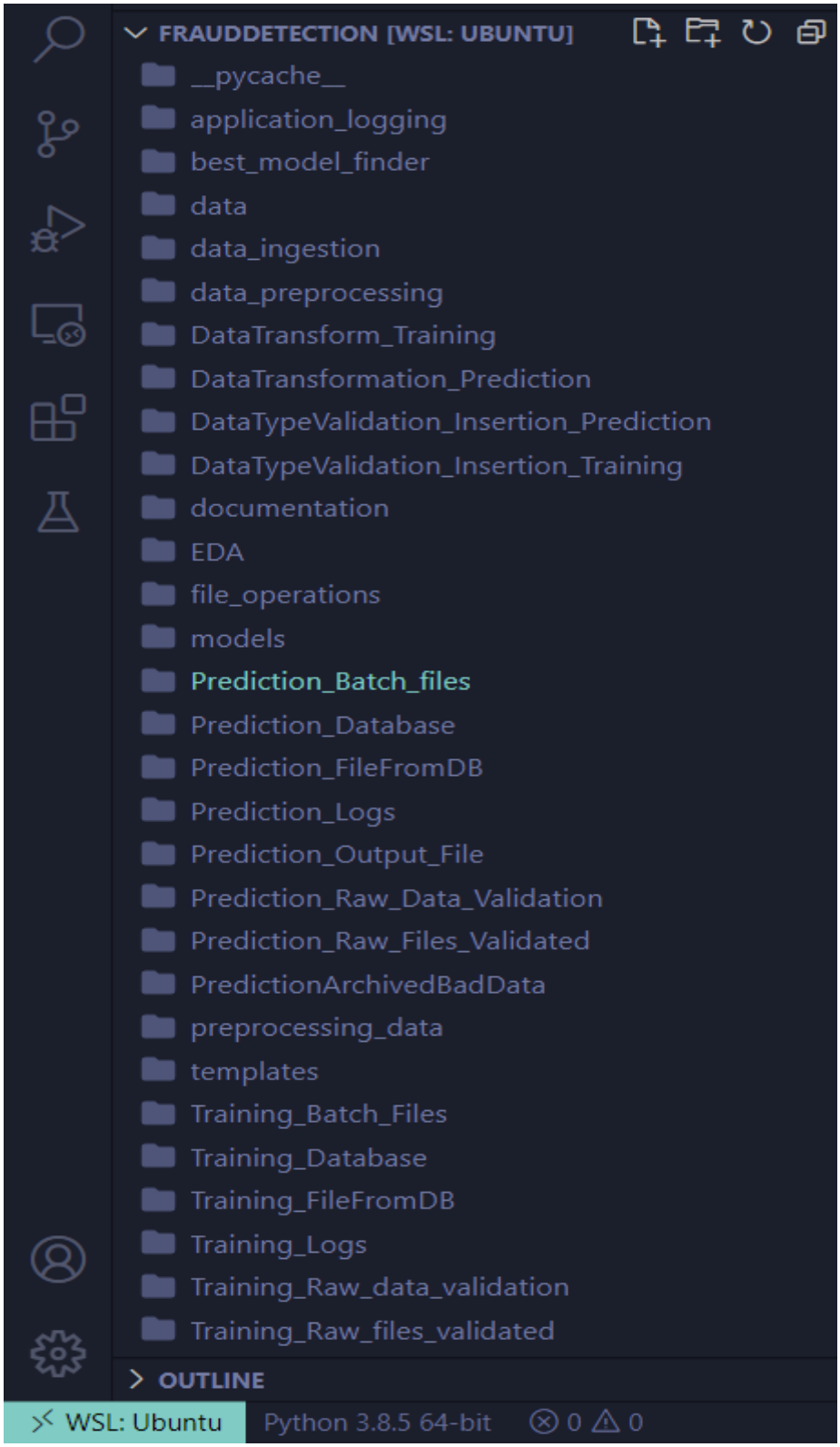
5) Once the prediction is made for all the clusters, the predictions along with the Wafer names are saved in a CSV file at a given location, and the location is returned to the client.

Deployment

We will be deploying the model to the Pivotal Cloud Foundry platform.

This is a workflow diagram for the prediction of using the trained model.

Now let's look at the project folder structure:



Steps before cloud deployment:

We need to change our code a bit so that it works unhindered on the cloud, as well.

a) Add a file called 'Procfile' inside the 'reviewScrapper' folder. This folder contains the command to run the flask application once deployed on the server:

```
web: gunicorn app:app
```

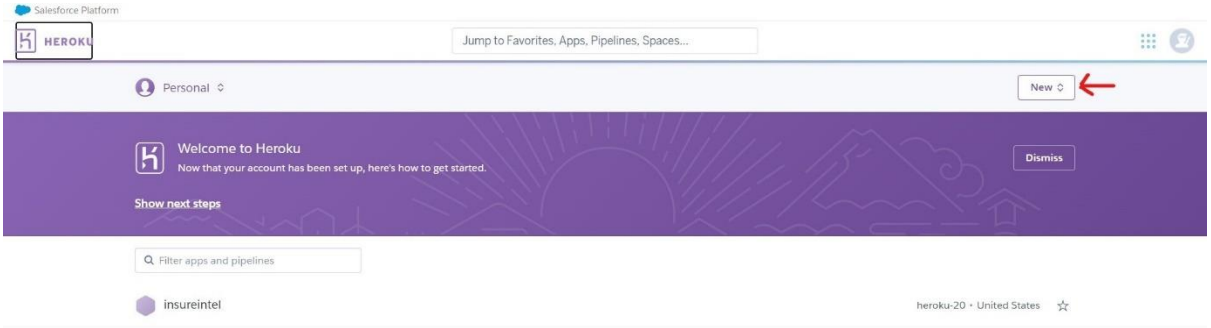
Here, the keyword 'web' specifies that the application is a web application. And the part 'app:app' instructs the program to look for a flask application called 'app' inside the 'app.py' file. Gunicorn is a Web Server Gateway Interface (WSGI) HTTP server for Python.

b) Open a command prompt window and navigate to your 'reviewScrapper' folder. Enter the command 'pip freeze > requirements.txt'. This command generates the 'requirements.txt' file.

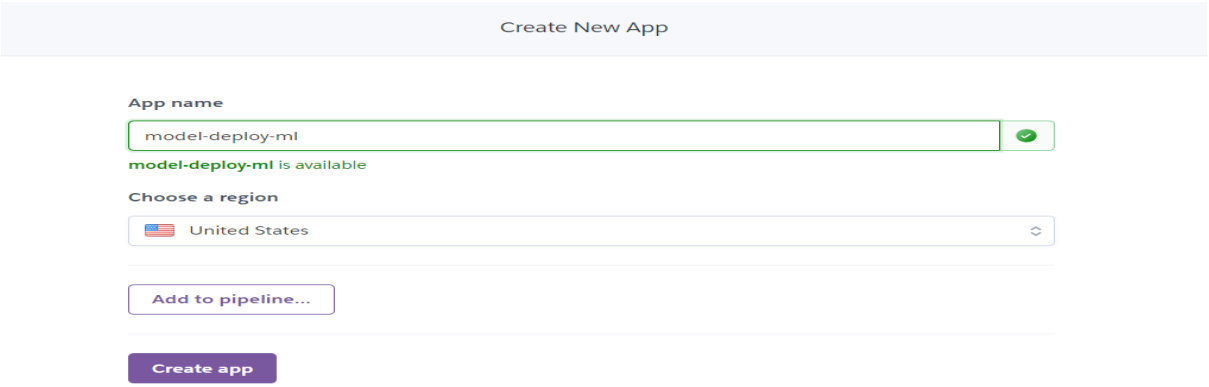
requirements.txt helps the Heroku cloud app to install all the dependencies before starting the webserver.

Deployment to Heroku:

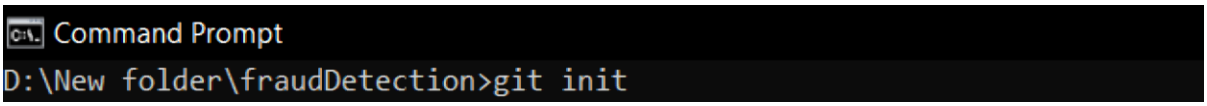
- Go to Heroku.com and create an account and login.
- Click on new(inside the red box) to create a new app.



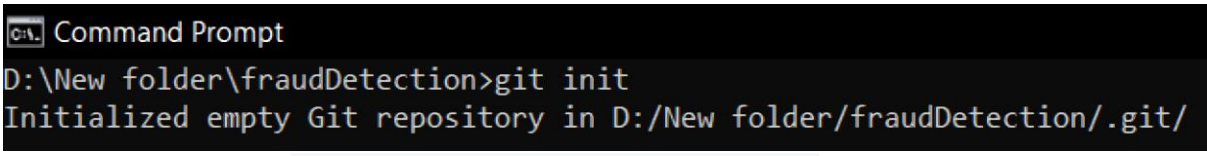
- Give the name of the app and click 'create app'.



- After app creation, the 'deploy' section has all the deployment steps mentioned.
- We'll download and install the Heroku CLI from the Heroku website: https://devcenter.heroku.com/articles/heroku-cli.
- Git also needs to be installed in your computer.
- After installing the Heroku CLI, Open a command prompt window and navigate to your project folder.
- Type the command `'heroku login'` to login to your heroku account as shown below:



  It opens up a webpage to login to Heroku.
- Before deploying the code to the Heroku cloud, we need to commit the changes to the local git repository.
- Type the command `'git init` to initialize a local git repository as shown below:



- Enter the command `heroku git:remote <remote repo name>` to connect local git repo to the remote repo.
- Enter the command `'git status'` to see the uncommitted changes
- Enter the command '`git add .'` to add the uncommitted changes to the local repository.
- Enter the command '`git commit -am "make it better"`' to commit the changes to the local repository.

- Enter the command '`git push heroku master`' to push the code to the heroku cloud.
- After deployment, heroku gives you the URL to hit the web API.

Once your application is deployed successfully, enter the command '`heroku logs --tail`' to see the logs.