

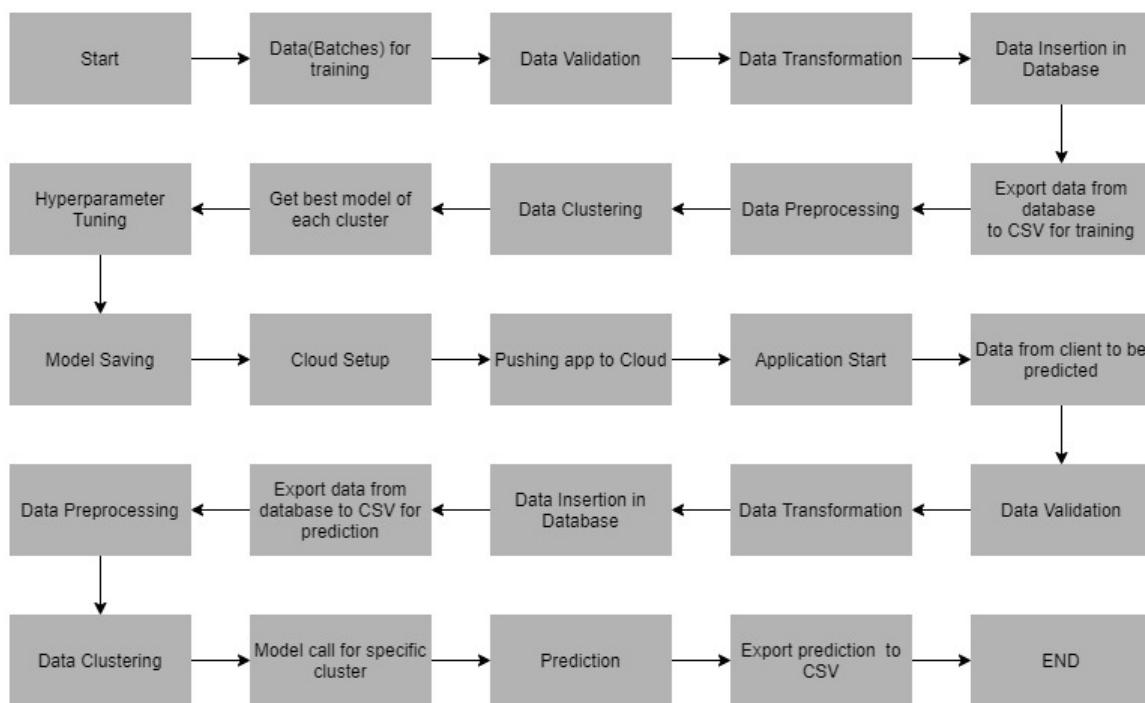
Project Title: Fitbit(Calories Burnt) Prediction.

Author: Oluwaseun Tope

Problem Statement

To build a regression model to predict the calories burnt based on the given indicators in the training data.

Architecture



Data Description

Data Description: This dataset predicts the visibility distance based on the different indicators as below:

1. Id: The customer ID
2. ActivityDate: The date for which the activity is getting tracked.

3. TotalSteps: Total Steps taken on that day.
4. TotalDistance: Total distance covered.
5. TrackerDistance: Distance as per the tracker
6. LoggedActivitiesDistance: Logged
7. VeryActiveDistance: The distance for which the user was the most active.
8. ModeratelyActiveDistance: The distance for which the user was moderately active.
9. LightActiveDistance: The distance for which the user was the least active.
10. SedentaryActiveDistance: The distance for which the user was almost inactive.
11. VeryActiveMinutes: The number of minutes for the most activity.
12. FairlyActiveMinutes: The number of minutes for moderately activity.
13. LightlyActiveMinutes: The number of minutes for the least activity
14. SedentaryMinutes: The number of minutes for almost no activity
15. Calories(Target): The calories burnt.

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. 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".

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".

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.

- d) Check for null values in the columns. If present, impute the null values
 - e) Scale the training and test data separately
- 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 two algorithms, "Decision Tree Regressor" and "XGBoost regressor". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the Rsquared 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

Client will send the data in multiple set of files in batches at a given location. Data will contain climate indicators in 10 columns.

Apart from prediction files, we also require a "schema" file from 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 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".

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".

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) Check for null values in the columns. If present, impute the null values.
 - d) Scale the training data.
- 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 original names before label encoder 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 Web Services Platform.

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

Now let's see the fitbit project folder structure.

Author: Oluwaseun Tope Oyero

The screenshot shows a file explorer window with the following project structure:

- fitbit (D:\datascience\iNeuron\MLProjects\fitbit)
 - application_logging
 - best_model_finder
 - data_ingestion
 - data_preprocessing
 - DataTransform_Training
 - DataTransformation_Prediction
 - DataTypeValidation_Insertion_Prediction
 - DataTypeValidation_Insertion_Training
 - EDA
 - EncoderPickle
 - file_operations
 - models
 - Prediction_Batch_files
 - Prediction_Database
 - Prediction_FileFromDB
 - Prediction_Logs
 - Prediction_Output_File
 - Prediction_Raw_Data_Validation
 - Prediction_Raw_Files_Validated
 - PredictionArchivedBadData
 - preprocessing_data
 - templates
 - Training_Batch_Files
 - Training_Database
 - Training_FileFromDB
 - Training_Logs
 - Training_Raw_data_validation
 - Training_Raw_files_validated
 - TrainingArchiveBadData
 - flask_monitoringdashboard.db
- main.py
- predictFromModel.py
- prediction_Validation_Insertion.py
- Problem Statement.docx
- requirements.txt

requirements.txt file consists of all the packages that you need to deploy the app in the cloud.

```
def predictRouteCurrent():
    try:
        if request.json is not None:
            path = request.json['filepath']

            pred_val = pred_validation(path) #object initialization

            pred_val.prediction_validation() #calling the prediction validation function

            pred = prediction(path) #object initialization

            # predicting for dataset present in database
            path = pred.predictionFromModel()
            return Response("Prediction File created at %s!!!" % path)
        elif request.form is not None:
            path = request.form['filepath']

            pred_val = pred_validation(path) #object initialization

            pred_val.prediction_validation() #calling the prediction validation function

            pred = prediction(path) #object initialization

            # predicting for dataset present in database
            path = pred.predictionFromModel()
            return Response("Prediction File created at %s!!!" % path)
```

main.py is the entry point of our application, where the flask server starts.

Author: Oluwaseun Tope Oyero

```
self.log_writer = logger.App_Logger()
self.pred_data_val = Prediction_Data_validation(path)

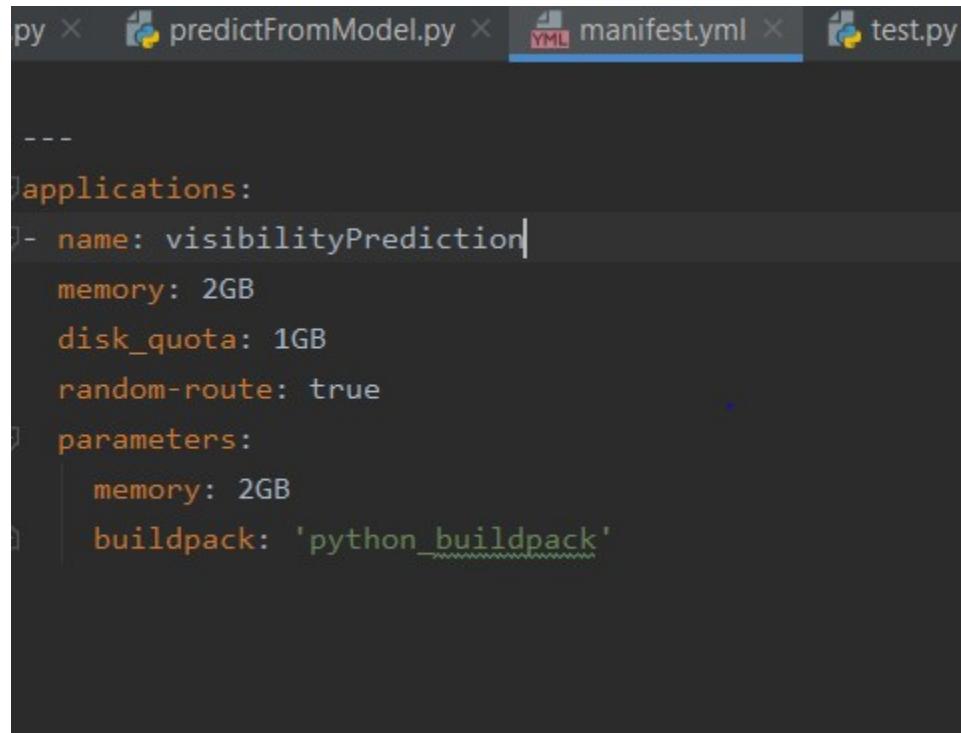
def predictionFromModel(self):

    try:
        self.pred_data_val.deletePredictionFile() #deletes the existing prediction file from last run!
        self.log_writer.log(self.file_object,'Start of Prediction')
        data_getter= data_loader_prediction.Data_Getter_Pred(self.file_object,self.log_writer)
        data=data_getter.get_data()

        #code change
        # wafer_names=data['Wafer']
        # data=data.drop(labels=['Wafer'],axis=1)

        preprocessor=preprocessing.Preprocessor(self.file_object,self.log_writer)
        data = preprocessor.dropUnnecessaryColumns(data,['DATE','Precip','WETBULBTEMPE','DewPointTempF','StationPressure'])
```

This is the **predictionFromModel.py** file where the predictions take place based on the data we are giving input to the model.



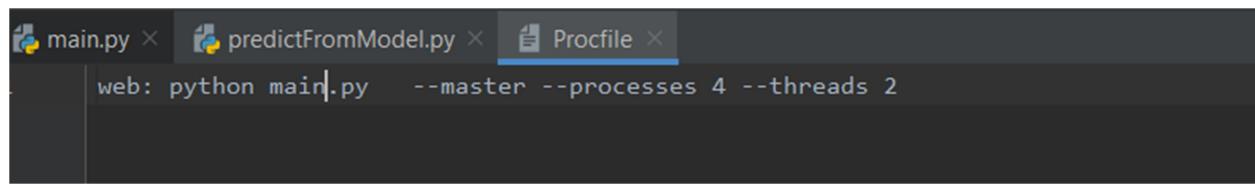
The screenshot shows a code editor interface with three tabs open: `manifest.yml`, `predictFromModel.py`, and `test.py`. The `manifest.yml` tab is active and displays the following YAML configuration:

```
---
```

```
applications:
  - name: visibilityPrediction
    memory: 2GB
    disk_quota: 1GB
    random-route: true
  parameters:
    memory: 2GB
    buildpack: 'python_buildpack'
```

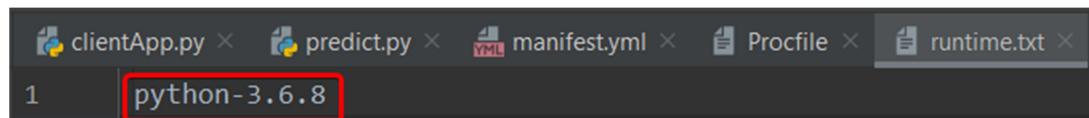
manifest.yml:- This file contains the instance configuration, app name, and build pack language.

Author: Oluwaseun Tope Oyero



```
main.py × predictFromModel.py × Procfile ×
web: python main.py --master --processes 4 --threads 2
```

Procfile :- It contains the entry point of the app.



```
clientApp.py × predict.py × manifest.yml × Procfile × runtime.txt ×
1 python-3.6.8
```

runtime.txt:- It contains the Python version number.