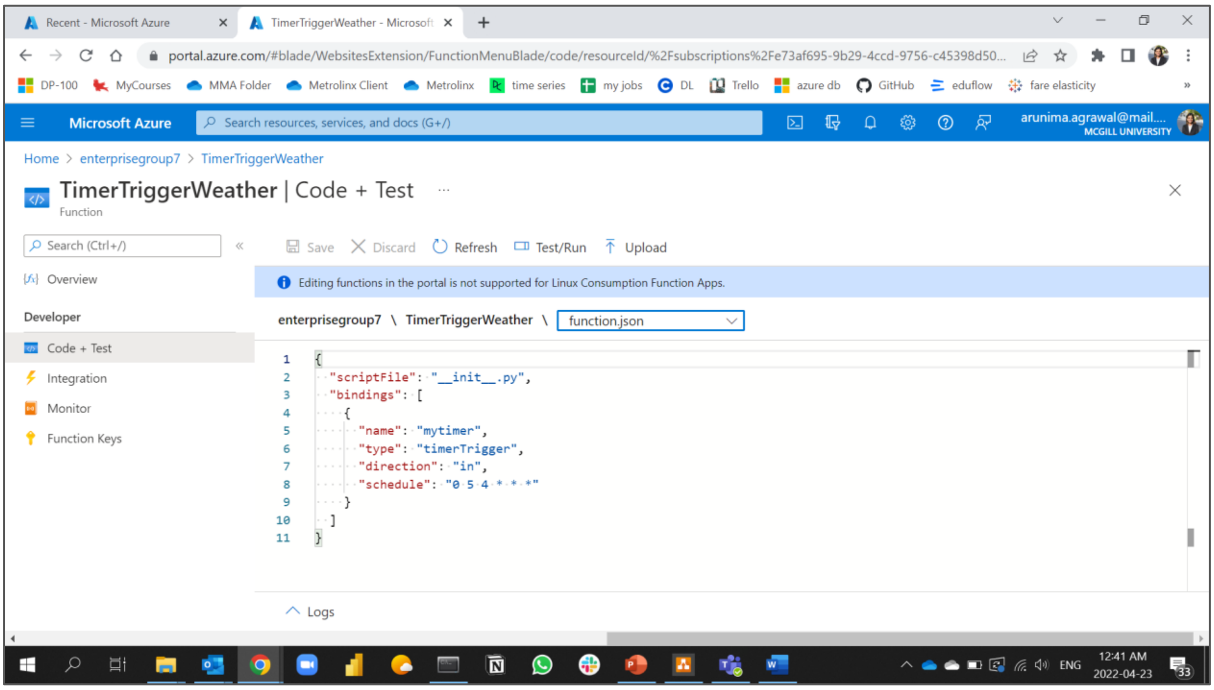
Enterprise II Screenshots – Group 7

Contents

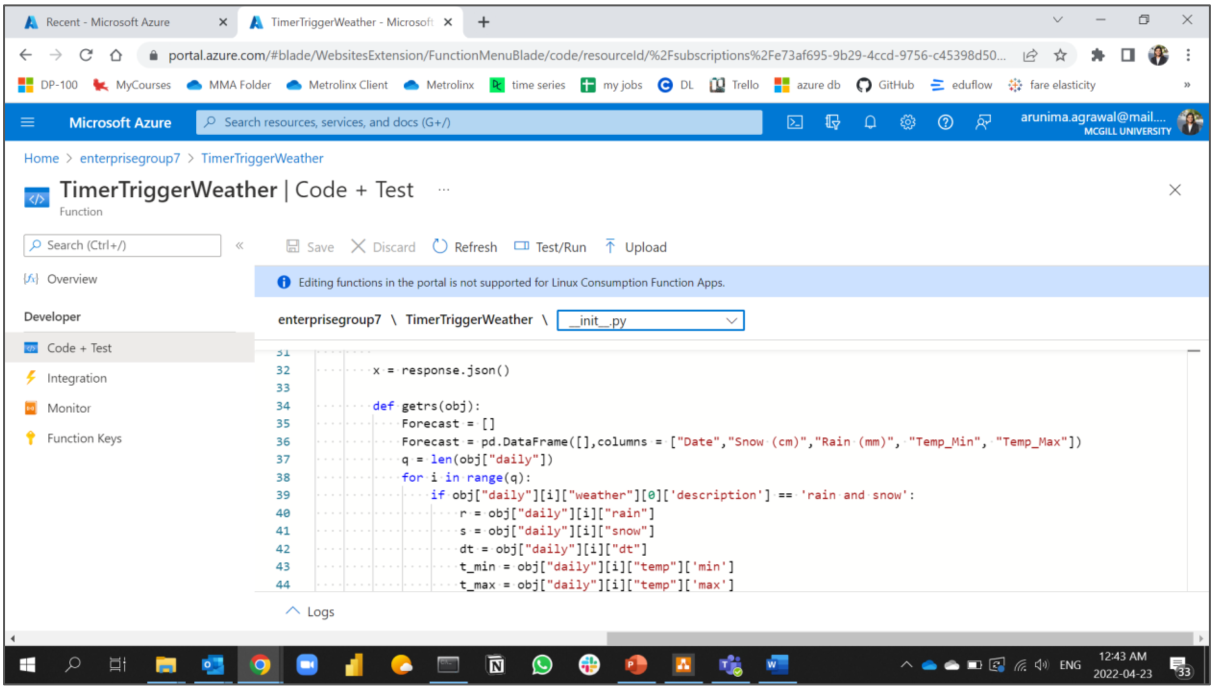
Azure Functions to fetch weather data using Open Weather API

Function scheduled to trigger at CRON schedule: “0 5 4 * * *”

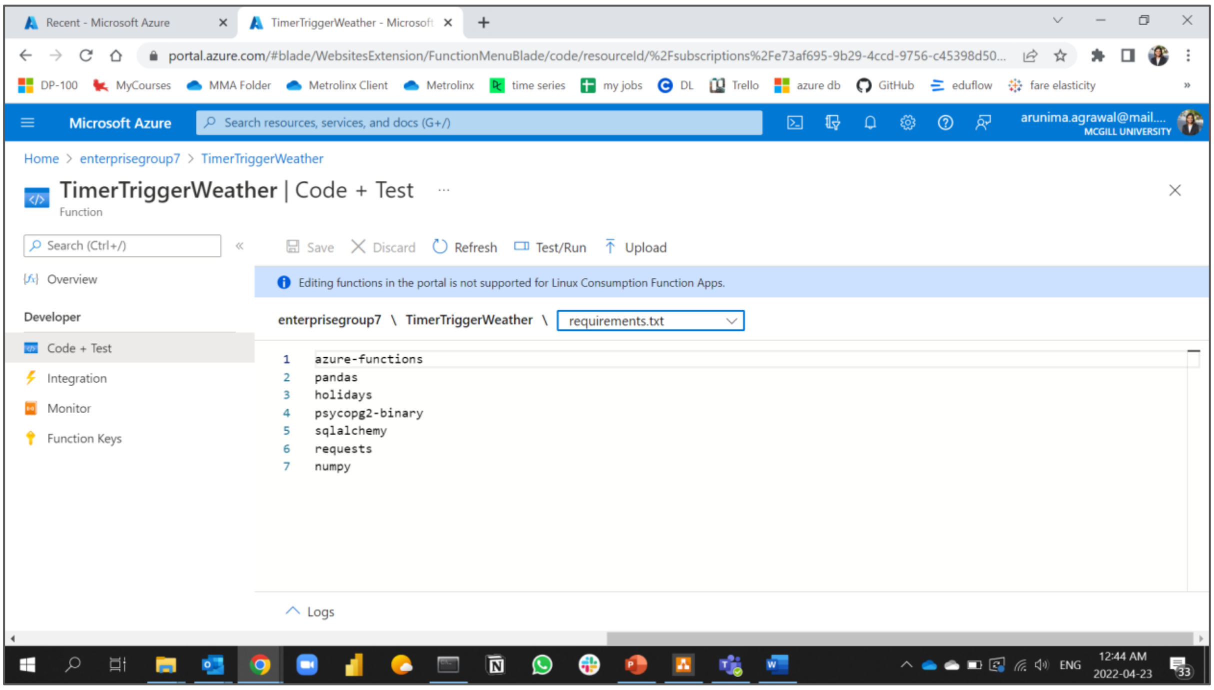
This means 4:05 AM UTC every day, or 12:05 AM EST every day.



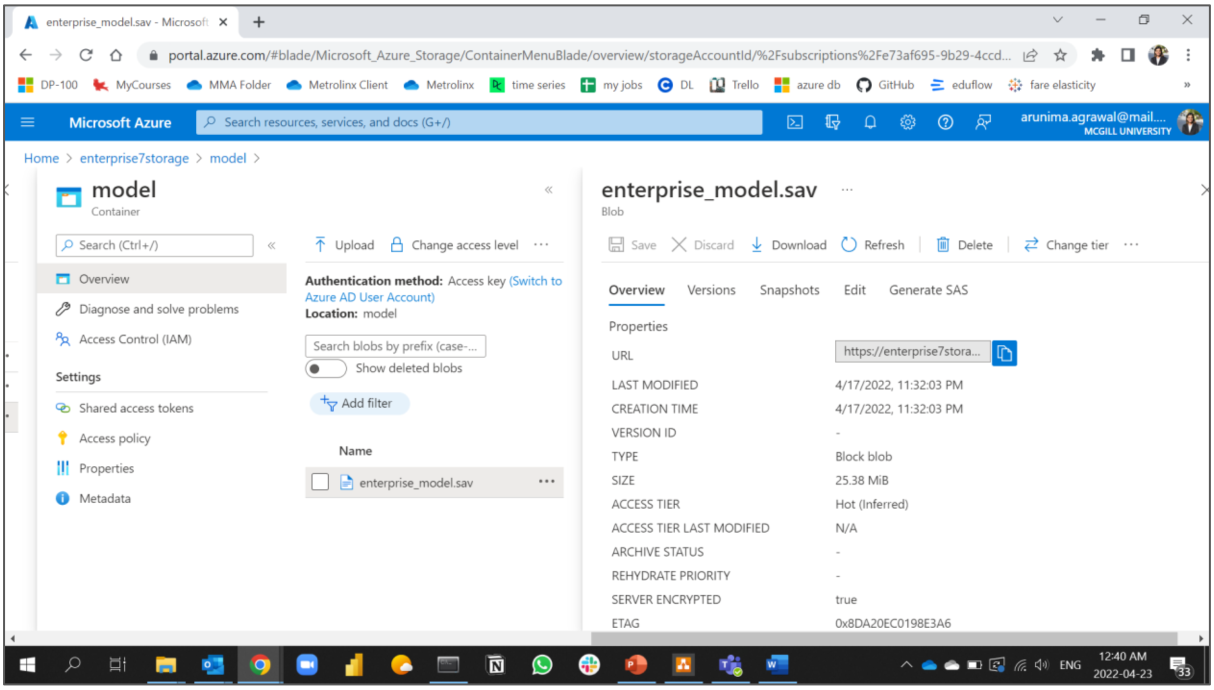
Python file that runs with the function trigger, and calls the Open Weather API to fetch weather data.



Requirements file containing the required PyPi packages required to run.



Azure Storage to store the model selected using TPOT



Azure PostgreSQL storing weather data, test data, model results

Table containing weather data generated using function trigger

In the below weather data, the “updated at” column shows the datetime when the function ran and the weather data last got updated.

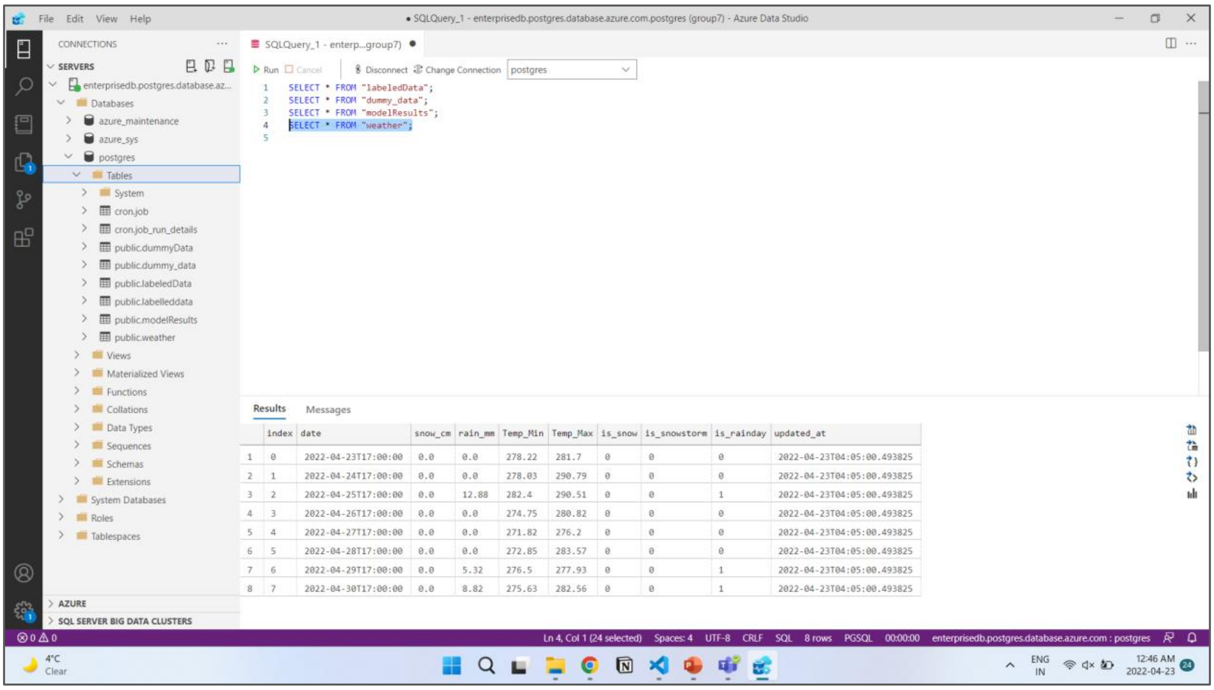
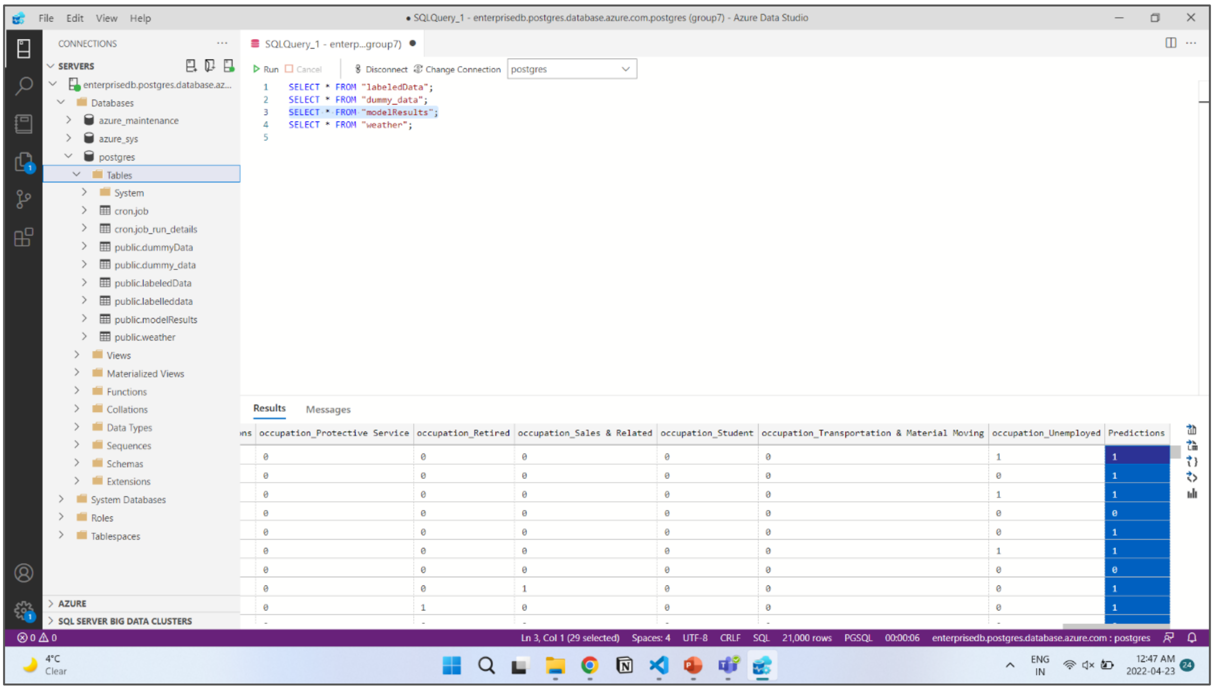


Table containing model results



Hyperparameter Optimization

1. Using Hyperopt
   Using the below space of parameters, we ran Bayesian optimization with hyperopt on Random Forest Classifier.

```python
#Space set close to values estimated by TPOT classifier
space = {'criterion': hp.choice('criterion', ['entropy', 'gini']),
        'bootstrap': hp.choice('bootstrap', [True, False]),
        'max_features': hp.choice('max_features', [0.3, 0.4, 0.5, 0.6]),
        'min_samples_split' : hp.choice ('min_samples_split', [7, 8, 9, 10, 11]),
        'n_estimators' : hp.choice('n_estimators', [150, 200, 250, 300, 350])
    }
space
```

```
{'criterion': <hyperopt.pyll.base.Apply at 0x2d831538340>,
 'bootstrap': <hyperopt.pyll.base.Apply at 0x2d831538460>,
 'max_features': <hyperopt.pyll.base.Apply at 0x2d8315385b0>,
 'min_samples_split': <hyperopt.pyll.base.Apply at 0x2d8315387f0>,
 'n_estimators': <hyperopt.pyll.base.Apply at 0x2d831538a00>}
```

```python
def objective(space):
    model = RandomForestClassifier(criterion = space['criterion'], bootstrap = space['bootstrap'],
                                   max_features = space['max_features'],
                                   #min_samples_leaf = space['min_samples_leaf'],
                                   min_samples_split = space['min_samples_split'],
                                   n_estimators = space['n_estimators'],
                                   )

    accuracy = cross_val_score(model, X_train, y_train, cv = 5).mean()

    # We aim to maximize accuracy, therefore we return it as a negative value
    return {'loss': -accuracy, 'status': STATUS_OK }
```

2. Using optuna

Optuna allowed us to further widen the space for hyper parameters for tuning.

```python
def objective(trial):
    min_samples_split = trial.suggest_int("min_samples_split", 7, 11)
    n_estimators = trial.suggest_int("n_estimators", 150, 350)
    max_features = trial.suggest_float("max_features", 0.3, 0.9)
    criterion = trial.suggest_categorical("criterion", ['entropy', 'gini'])
    bootstrap = trial.suggest_categorical("bootstrap", [True, False])

    rf_model = RandomForestClassifier(
        criterion = criterion,
        bootstrap = bootstrap,
        max_features = max_features,
        min_samples_split = min_samples_split,
        n_estimators = n_estimators)
    score = cross_val_score(rf_model, X, y,  cv=5).mean()
    return score
```

```python
study = optuna.create_study(direction = "maximize")
study.optimize(objective, n_trials = 10)
trial = study.best_trial
```

Auto ML

1. TPOT

```
tpot = TPOTClassifier(verbosity=2, max_time_mins=30, scoring='accuracy')
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
```

```
Optimization Progress:   0%|          | 0/100 [00:00<?, ?pipeline/s]
31.88 minutes have elapsed. TPOT will close down.
TPOT closed during evaluation in one generation.
WARNING: TPOT may not provide a good pipeline if TPOT is stopped/interrupted in a early generation.


TPOT closed prematurely. Will use the current best pipeline.

Best pipeline: RandomForestClassifier(Normalizer(input_matrix, norm=max), bootstrap=True, criterion=gini, max_features=0.5, min_samples_leaf=1, min_samp
les_split=9, n_estimators=100)
0.7591643673630272
```

Using TPOT to find best estimator pipeline based on training data.
Due to limited computational resource, limiting the estimator time to 30 mins and using the best estimator recommended within the stipulated run.

2. H2o.ai

```
h2o_aml = H2OAutoML(max_models = 10, seed = 1, exclude_algos = ["StackedEnsemble"], verbosity="info") #max_runtime_secs=120,
```

```
h2o_aml.train(x = x, y = y, training_frame = train)
```

```
17:23:17.888: AutoML: starting GBM_5_AutoML_1_20220417_164838 model training
17:23:17.888: _train param, Dropping bad and constant columns: [toCoupon_GEQ5min]


17:23:29.94: AutoML: starting DeepLearning_1_AutoML_1_20220417_164838 model training
17:23:29.97: _train param, Dropping bad and constant columns: [toCoupon_GEQ5min]


17:23:36.244: Skipping StackedEnsemble 'best_of_family_3' due to the exclude_algos option or it is already trained.
17:23:36.245: Skipping StackedEnsemble 'all_3' due to the exclude_algos option or it is already trained.
17:23:36.249: AutoML: starting GBM_grid_1_AutoML_1_20220417_164838 hyperparameter search

█| (done) 100%

17:23:50.407: Skipping StackedEnsemble 'best_of_family_4' due to the exclude_algos option or it is already trained.
17:23:50.407: Skipping StackedEnsemble 'all_4' due to the exclude_algos option or it is already trained.
17:23:50.407: Skipping StackedEnsemble 'best_of_family_5' due to the exclude_algos option or it is already trained.
17:23:50.407: Skipping StackedEnsemble 'all_5' due to the exclude_algos option or it is already trained.
```

Training H2O on training data to find the best possible estimator.
Running it to get top 10 models, skipping stacked ensemble algos to reduce computational resource strain.

Synthetic Data Generation (used as Test Data)

Following is the Bayesian Network created for the generation of the synthetic data by the describer object with the following hyperparameter:

1. thresholdthreshold_value = 30 (An attribute is categorical if its domain size is less than this)
2. epsilon = 1 (A parameter in Differential Privacy i.e., noise addition in the synthetic data)
3. degree_of_bayesian_network = 2 (The maximum number of parents in Bayesian network, i.e., the maximum number of incoming edges)
4. num_tuples_to_generate = 21000 (Number of tuples generated in synthetic dataset)

```
========================== BN constructed ==========================
Constructed Bayesian network:
    income               has parents ['Bar'].
    occupation           has parents ['income', 'Bar'].
    age                  has parents ['occupation', 'income'].
    education            has parents ['age', 'occupation'].
    has_children         has parents ['age', 'occupation'].
    CarryAway            has parents ['age', 'occupation'].
    maritalStatus        has parents ['has_children', 'education'].
    toCoupon_GEQ25min    has parents ['has_children', 'income'].
    direction_opp        has parents ['toCoupon_GEQ25min', 'maritalStatus'].
    direction_same       has parents ['direction_opp', 'occupation'].
    Restaurant20To50     has parents ['maritalStatus', 'income'].
    CoffeeHouse          has parents ['occupation', 'income'].
    gender               has parents ['direction_opp', 'toCoupon_GEQ25min'].
    Y                    has parents ['occupation', 'Bar'].
    destination          has parents ['direction_opp', 'has_children'].
    time                 has parents ['destination', 'Bar'].
    passanger            has parents ['destination', 'Bar'].
    toCoupon_GEQ15min    has parents ['passanger', 'income'].
    RestaurantLessThan20 has parents ['CoffeeHouse', 'direction_same'].
    toCoupon_GEQ5min     has parents ['passanger', 'CarryAway'].
    expiration           has parents ['passanger', 'time'].
    coupon               has parents ['toCoupon_GEQ5min', 'CoffeeHouse'].
```

Following is the pairwise comparison between Original and Synthetic dataset generated i.e., pairwise correlation analysis to replicate the original dataset in correlation mode by the Bayesian Network