Configuration Manual

MSc Research Project
Cloud Computing

Sruthi Praveen
Student ID: x21223785

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sruthi Praveen
Student ID:	x21223785
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	18/08/2023
Project Title:	Configuration Manual
Word Count:	1054
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sruthi Praveen
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sruthi Praveen
x21223785

1 Introduction

The configuration file covers the critical dependencies needed for the project "Cloud-Optimized Fusion of Time Series and Machine Learning Models for Enhanced Real-Time Forecasting." The file comprises of a carefully curated set of critical Python modules that enable different features that are critical to the project's success. These libraries encompass a range of tasks, including web application development, data manipulation, numerical computation, visualization, statistical modeling, machine learning, and specialized time series forecasting. With the help of this configuration file, developers can easily set up their environment and gain access to the resources required to create a dynamic and robust forecasting system that uses both time series and machine learning approaches.

2 Software Tools and Libraries Required

The following software tools are required for successful implementation of the project:

1. Visual Studio Code - Visual Studio Code - offers a solid programming environment for smooth Python integration, making it a great choice for Python-based implementation.

2. Postman - provides essential API testing capabilities for validating the Flask API

3. Python - used for data analysis, model creation, and several other tasks in the project.

4. Flask - It is a Python web framework that enables for the rapid development of web-based applications. It offers tools and libraries for developing web APIs, managing routes, templates, and managing server-side functionality. Flask is used in this project to develop a web service that can publish requests, allowing forecasting models to be integrated with cloud infrastructure.

5. pm2 - PM2 is a Node.js application process manager. It is used to manage and deploy applications, assuring that they function in the background reliably. In this project, PM2 is used to launch and maintain the Flask application.

6. pandas - Pandas is a strong Python data manipulation toolkit. It includes data structures and tools for manipulating, analyzing, and visualizing data. Pandas aids us in working with structured data in this project, such as time series and datasets, by making it easier to preprocess and manipulate the data before feeding it to the forecasting models.

7. numpy - NumPy is the primary Python library for scientific computing. It allows us to do efficient numerical operations by supporting arrays, matrices, and mathematical functions. NumPy is used in this project to conduct numerical computations that are required for data pre processing and other computations needed by the forecasting models.

8. matplotlib - Matplotlib is a Python library that may be used to produce static, interactive, and animated visualizations in applications. It is used in this project to create graphical representations of time series data, forecast outcomes, and model evaluations, which aid in the comprehension of the findings.

9. statsmodels - Statsmodels is a statistical model building and hypothesis testing package. It includes a number of tools for estimating and analyzing statistical models, including time series models. Here, it is used to create and analyze time series models like ARIMA, which are critical for projecting future values based on historical data patterns.

10. scikit-learn - It offers simple and effective data mining and data analysis capabilities. Scikit-learn is used in this project to train, evaluate, and deploy machine learning models that complement the time series models, resulting in improved forecasting accuracy.

11. sktime - It is a Python-based time series forecasting package providing a comprehensive range of time series analysis and forecasting tools and algorithms. Sktime includes specific forecasting techniques such as AutoARIMA and STLForecaster in this project, allowing us to use complex time series forecasting methodologies.

3 Software Installation

```
pip install Flask
pip install pandas
pip install numpy
pip install matplotlib
pip install statsmodels
pip install scikit-learn
pip install sktime
npm install pm2 -g
pm2 start "python3 flask_code.py" --name "timeseries"
```

Figure 1: Library Installation

1. Installation of the required libraries are done with the following commands:[Fig. 1]

pip install Flask - This command installs the Flask library, which is used to build the API.

pip install pandas - installs the pandas data manipulation and analysis library.

pip install numpy - installs the numpy numerical calculation library.

pip install matplotlib -installs the Matplotlib library for data plotting and visualization of data

pip install statsmodels - installs the Statsmodels library for estimating and understanding statistical models.

pip install scikit-learn - installs the scikit library, which can be used for classification, regression, clustering, dimensionality reduction, model selection, and other tasks.

pip install sktime - This command installs the sktime library, which is used for time series forecasting, classification, and regression.

npm install pm2 -g - Installs PM2 for Node.js application management and deployment.

pm2 start "python3 flask_code.py" –name "timeseries" - starts the Flask application with PM2.

2. Then installed libraries are then imported as shown in Fig.3

```
from flask import Flask, request, jsonify
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
import json
import math
import copy
import os
import math
from math import sqrt
import matplotlib.pyplot as plt
import statistics
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import LassoLars
from sklearn.linear_model import PassiveAggressiveRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import ExtraTreeRegressor
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

Figure 2: Library Import

```python
from sklearn.tree import ExtraTreeRegressor
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sktime.forecasting.arima import AutoARIMA
from scipy.stats import randint as sp_randint
from sktime.forecasting.trend import STLForecaster
# from IPython import get_ipython
import logging
import warnings
warnings.filterwarnings('ignore')
```

Figure 3: Library Import

4 Implementation

• Data is processed and transformed based on different flags that indicate the type of processing required. [Fig. 4]

```python
def assign_dates(data, flag, dates='', sku=""):  # changed
    if flag == 'validation':
        dates = dates.reset_index()
        dates.columns = ['time', 'sales']
        dates.time = pd.to_datetime(
            dates.time, format='%m/%d/%y', infer_datetime_format=True)
        dates.time = dates.time.dt.to_period('M')
        data = [float(format(i, '.3f')) for i in data]
        result = pd.DataFrame(
            {'time': dates.time.astype(str), 'validation': data})
        result.set_index('time', inplace=True)
        result = result.to_dict()
        result = result['validation']

    elif flag == 'val_facc':
        dates = dates.reset_index()
        dates.columns = ['time', 'sales']
        dates.time = pd.to_datetime(
            dates.time, format='%m/%d/%y', infer_datetime_format=True)
        dates.time = dates.time.dt.to_period('M')
        result = pd.DataFrame(
            {'time': dates.time.astype(str), 'val_facc': data})
        result.set_index('time', inplace=True)
        result = result.to_dict()
        result = result['val_facc']

    elif flag == 'forecast':
        dates = dates.reset_index()
        dates.columns = ['time', 'forecast']
        last_date = pd.to_datetime(
            dates.time[0], format='%m/%d/%y', infer_datetime_format=True)
        # print(last_date,forecast_period,type(forecast_period))
        date_range = pd.date_range(last_date, periods=int(
            12), freq='M')  # changed1st
        date_range = date_range.strftime('%Y-%m').tolist()
        poped_date = date_range.pop(0)  # same month as last_date, FUSO
        data = [float(format(i, '.3f')) for i in data]
```

Figure 4: Data Pre Processing

• Calculating forecast accuracy for specific SKUs in the time series data set.[Fig. 5]

• Outlier treatment is performed on the dataset using a sliding window approach with a specified interval and partition size. [Fig. 6]

• Supervised dataset is created after data pre processing making it suitable for regression tasks.[Fig. 7]

• Making predictions using various machine learning models for time series forecasting.[Fig. 8]

• Model selection and hyperparameter optimization for the time series forecasting.[Fig.9]

• Selecting the appropriate modeling technique based on the input algorithm and order, and producing forecasted values.[Fig. 10]

• Defines an endpoint that accepts data uploads, processes the data to generate forecasts for different keys, and returns the forecasted results in JSON format.[Fig. 11]

```python
def calculate_forecast_accuracy(expected, forecast):
    if math.isnan(expected):
        expected = 0
    else:
        expected = int(expected)
    expected = int(expected)
    forecast = int(forecast)
    print("calculate_forecast_accuracy")
    facc = (1 - (np.abs(expected - forecast)) /
            (expected+(expected == 0))) * 100
    if facc < 0:
        facc = 0
    mape = (np.abs(expected - forecast) / expected) * 100
    bias = (expected - forecast)

    if np.isnan(facc) == True or np.isfinite(facc) == False:
        facc = 0
    if np.isnan(mape) == True or np.isfinite(mape) == False:
        mape = 0
    if np.isnan(bias) == True or np.isfinite(bias) == False:
        mape = 0
    return float(format(facc, '.3f')), float(format(mape, '.3f')), float(format(bias, '.3f'))
```

Figure 5: Forecast accuracy

```python
def outlier_treatment_tech(dataset, interval, pt):
    # TODO: remove interval
    start = 0
    end = pt
    sku_data = [0]*len(dataset)
    while end < len(dataset):
        sku_data[start:end] = outlier_treatment(dataset[start:end])
        start = end
        end += pt
    if start < len(dataset):
        sku_data[start:len(dataset)] = outlier_treatment(
            dataset[start:end])
#         sku_data=sku_data.append(outlier_treatment(dataset[start:len(dataset)]))
    sku_data = pd.DataFrame(sku_data)
    return sku_data

Run Cell | Run Above | Debug Cell
# In[18]:

def acf_plot(dataset, freq):
    res = acf(dataset)
    # plot_acf(dataset)
    acfval = [0]*len(res)
    ub = 1.96/np.sqrt(len(dataset))
    p1 = 1
    for i in range(len(res)):
        acfval[i] = abs(res[i])
    acfval.sort(reverse=True)
    acfval = np.array(acfval[1:])
    pshort = np.array(acfval[0:3])
    pshortind = [0]*len(pshort)
#     print(pshort)
    for i in range(len(pshort)):
        pshortind[i] = np.where(abs(res) == pshort[i])[0][0]
    ind = np.where(acfval > ub)[0]
    finalacf = acfval[ind]
    plist = [0]*len(finalacf)
```

Figure 6: Outlier Treatment

5 AWS Deployment

The models are then deployed in AWS Cloud.[Fig.12,13]

6 Postman Installation and File Import

1. Download the appropriate version of Postman depending on the operating system (Windows, macOS, or Linux).

2. Launch Postman once the installation is complete.

3. Click on "Import" button in the upper left corner.[Fig. 14]

4. In the URL field, enter the URl: `https://api.postman.com/collections/12533933-85644ef0-ac1a-4fb8-9986-e8256dc59d4b?access_key=PMAT-01H7H57YC0CRQWFE1FTA9T1`

```python
def timeseries_to_supervised(dataset, lag=1):
    dataset = pd.DataFrame(dataset)
    y = [dataset.shift(i) for i in range(1, lag + 1)]
    y.append(dataset)
    dataset = pd.concat(y, axis=1)
    cols = []
    for i in range(lag):
        cols.append('x_' + str(i))
    cols.append('y')
    dataset.columns = cols
#    print(dataset.columns)
    dataset.dropna(axis=0, inplace=True)
#    dataX, dataY = [], []
#    for i in range(len(dataset)-look_back):
#        a = dataset[i:(i + look_back),0]
#        dataX.append(a)
#        dataY.append(dataset[i + look_back,0])
#    return np.array(dataX), np.array(dataY)
    return dataset

def scaler_selection(key):
    if key == 'lr' or key == 'lasso' or key == 'ridge' or key == 'knn':
        scaler = MinMaxScaler(feature_range=(0, 1), copy=True)
    elif key == 'svmr':
        scaler = StandardScaler()

    return scaler
```

Figure 7: Creating a supervised dataset

```python
def model_ML(dataset=[], tsize=0, test_shape=0, model=np.nan, key='', order=(0, 0, 0), train_flag=0):
    predictions = []
    pred_temp = []
    rmse_val = []
    scale_flag = 0
    # print(dataset)
    #print(1)
    if key == 'lr' or key == 'lasso' or key == 'ridge' or key == 'knn' or key == 'svmr':
        scale_flag = 1

    if train_flag == 1:
        itr = 12 # changed
        #print(3)
        for i in range(3):
            expected = pd.DataFrame(dataset)
            expected = expected.tail(itr).head(3)
            expected = expected.reset_index(drop=True)
            # print(dataset[:-itr])
            train = dataset[:-itr]

            diff_values = difference(train, order[1])

            if scale_flag == 1:
                scaler = scaler_selection(key)
                diff_values = scaler.fit_transform(
                    pd.DataFrame(diff_values).values.reshape(-1, 1))
            #print(4)
            supervised = timeseries_to_supervised(train, order[0])

            data = supervised.values
            #print("fit")
            RF_model = fit_model(data, model)
            pred_temp = []
            #print("data")
            for j in range(test_shape):
                X = data[:, 0:-1]
                yhat = forecast_model(RF_model, X)
```

Figure 8: ML Models

```python
def models_ML():
    models = dict()
    n_trees = 100
    # prameters for RandomSearch
    lr_param = {"fit_intercept": [True, False], "normalize": [
        False], "copy_X": [True, False]}
#    knn_param = {"n_neighbors":[2,3,4,5,6,7,8],"metric": ["euclidean", "cityblock"]}
    dtree_param = {"max_depth": [3, None], "min_samples_leaf": sp_randint(1, 11), "criterion": [
        "mse"], "splitter": ["best", "random"], "max_features": ["auto", "sqrt", None]}
    lasso_param = {"alpha": [0.02, 0.024, 0.025, 0.026, 0.03], "fit_intercept": [
        True, False], "normalize": [True, False], "selection": ["random"]}
    ridge_param = {"alpha": [200, 230, 250, 265, 270, 275, 290, 300, 500], "fit_intercept": [
        True, False], "normalize": [True, False], "solver": ["auto"]}
    elas_param = {"alpha": list(np.logspace(-5, 2, 8)), "l1_ratio": [.2, .4, .6, .8], "fit_intercept": [
        True, False], "normalize": [True, False], "precompute": [True, False]}
#    g = [pow(2,-15),pow(2,-14),pow(2,-13),pow(2,-12),pow(2,-11),pow(2,-10),pow(2,-9),pow(2,-8),pow(2,-7),pow(2,-6),pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2,-1),pow
#    c=[pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2,-1),pow(1,0),pow(2,1),pow(2,2),pow(2,3),pow(2,4),pow(2,5),pow(2,6),pow(2,7),pow(2,8),pow(2,9),pow(2,10),pow(2,11),p
#    svr_param={"C":c,"gamma":g,"kernel":["rbf","sigmoid"]}
#    gb_param={"n_estimators":[1, 2, 4, 8, 16, 32, 64, 100, 200],"max_depths":list(np.linspace(1, 32, 32, endpoint=True)),"min_samples_splits":list(np.linspace(0.1, 1.0,

    models['lr'] = RandomizedSearchCV(
        LinearRegression(), lr_param, n_jobs=1, random_state=42)
    models['lasso'] = RandomizedSearchCV(
        Lasso(), lasso_param, n_jobs=1, n_iter=100, random_state=42)
    models['ridge'] = RandomizedSearchCV(
        Ridge(), ridge_param, n_jobs=1, n_iter=100, random_state=42)
    models['en'] = RandomizedSearchCV(ElasticNet(
    ), elas_param, scoring='neg_mean_squared_error', n_jobs=1, n_iter=100, cv=10, random_state=42)
    #models['huber']             = HuberRegressor()
    models['llars'] = LassoLars()
    models['pa'] = PassiveAggressiveRegressor(
        max_iter=1000, tol=1e-3, random_state=42)
    models['knn'] = KNeighborsRegressor(n_neighbors=3)
    models['cart'] = RandomizedSearchCV(
        DecisionTreeRegressor(), dtree_param, n_jobs=1, n_iter=100, random_state=42)
    models['extra'] = ExtraTreeRegressor(random_state=42)
```

Figure 9: Model Selection

5. Click on the "Body" tab below the URL field.

6. Choose the "form-data" option and give the dataset, forecast type and forecast period.15

```python
def model_predict(best_algo, best_order, data, forecast_period, rept=0):

    predictions = []
    ML_models = models_ML()
    ARIMA_models = init_ARIMA_models(best_order)

    if best_algo in ML_models.keys():
        if best_algo == 'GR_LR':
            predictions = model_LinearRegression(
                data.values, forecast_period, best_order)
        elif best_algo == 'SVR_Sigmoid':
            predictions = model_SVR_Sigmoid(
                data.values, forecast_period, best_order)
        elif best_algo == 'SVR_RBF':
            predictions = model_SVR_RBF(
                data.values, forecast_period, best_order)
        else:
            test_shape_add = test_shape_adder(best_algo)
            test_shape_fin = forecast_period+test_shape_add
            predictions, rmse = model_ML(dataset=data.values, tsize=forecast_period,
                                         test_shape=test_shape_fin, model=ML_models[best_algo], order=best_order)
            if test_shape_add > 0:
                st = test_shape_fin-1
                end = test_shape_add-1
                predictions = predictions[-st:-end]

    elif best_algo in ARIMA_models.keys():
        #       print("lollllllllzzz",ARIMA_models[best_algo])
        predictions, rmse = model_ARIMA(
            best_algo, train=data.values, test_shape=forecast_period, order=ARIMA_models[best_algo], train_flag=0)

    elif best_algo in ['SES', 'HWES']:
        predictions, rmse = model_ES(
            best_algo, train=data.values, test_shape=forecast_period, train_flag=0)
    elif best_algo in ['naive', 'naive2', 'naive3', 'naive6', 'naive12', 'naive12wa', 'naive6wa']:
        predictions, rmse = model_Naive(
            best_algo, data.values, forecast_period, best_order, rept, train_flag=0)
```

Figure 10: Model Prediction

```python
@app.route('/forecast', methods=['POST'])
def forecast():
    file = request.files.get('file')
    forecast_type = request.form['forecast_type']
    forecast_period = int(request.form['forecast_period'])
    isFile = os.path.isfile('temp.csv')
    if isFile:
        os.remove('temp.csv')
    file.save('temp.csv')

        # Convert the data dictionary to a DataFrame

    df = pd.read_csv('temp.csv')
    a,b = df.shape
    keys_values = df['keys'].values
    Final_Forecast = pd.DataFrame()
    for num in keys_values:
        DataForecast = pd.DataFrame()
        data = df[df['keys']==num]
        t = list(data.columns.values)
        t.pop(0)
        val = data.pop("keys")
        d = data.T.values
        TSValues = []
        for num in d:
            TSValues.append(num)
        DataForecast['ACR'] = TSValues
            # Call the predict function to get the forecast
        Final_Forecast[list(val.values)[0]] = predict(DataForecast, int(forecast_period), forecast_type.lower())

        # Convert the forecast DataFrame to a JSON response
    forecast_json = Final_Forecast.to_json(orient='split')

    return jsonify({'forecast': forecast_json})


if __name__ == '__main__':
```

Figure 11: Flask Application

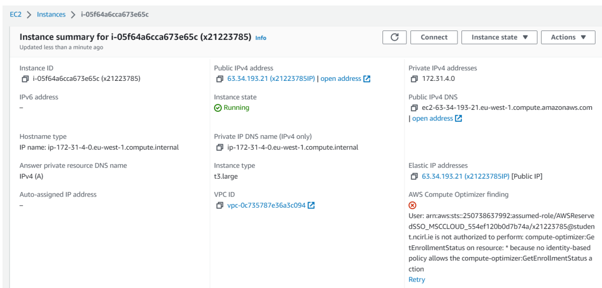

Figure 12: EC2 Instance

7. Click on ”Send” to view the forecasted results16.

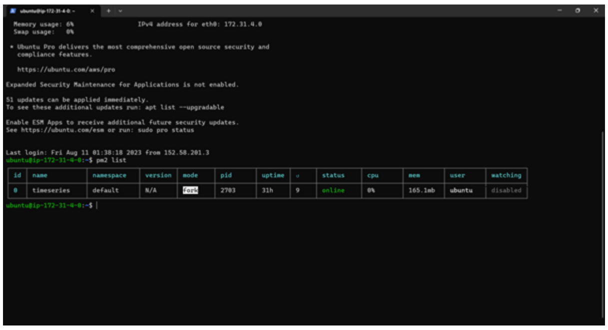

Figure 13: Application Status

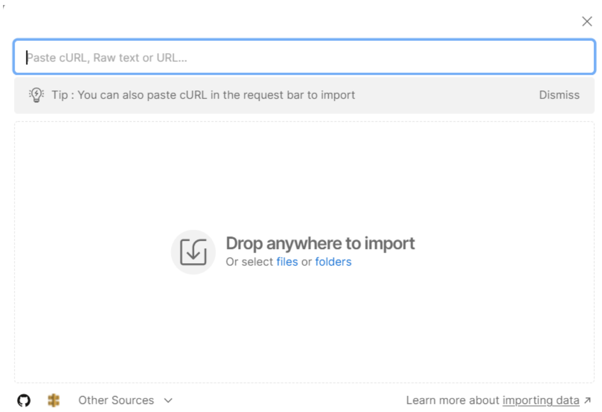

Figure 14: Import End Point URL

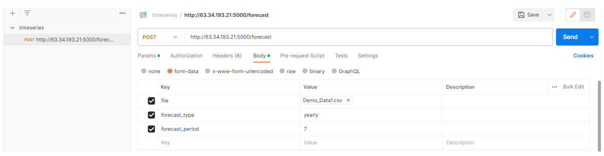

Figure 15: Input Parameters

```python
def output_forecast(sku, dataset, sku_data, output, forecast_results):
    dataset = dataset.reset_index()
    dataset.columns = ['time', 'sales']

    output['last_date'] = dataset.time.iloc[-1]

    forecast_result = add_forecasted_results(
        sku, dataset, sku_data, output)
    forecast_results.append(forecast_result)

    # with open(r'Forecasting_FRA_Dec.json', 'w', encoding = 'utf-8') as f:
    #     json.dump(forecast_results, f, ensure_ascii = False, indent = 4, default = str)
    #f = pd.DataFrame(forecast_results)
    return forecast_results

def add_forecasted_results(sku, dataset, data, output):
    sku_data = dict()
    sku_data['sku'] = sku
#    sku_data['dataset'] = dataset
#    sku_data['sku_data'] = data

    for key in output:
        sku_data[key] = output[key]

    return sku_data

Run Cell | Run Above | Debug Cell
# In[9]:

def calculate_forecast_accuracy(expected, forecast):
    if math.isnan(expected):
        expected = 0
    else:
        expected = int(expected)
    expected = int(expected)
    forecast = int(forecast)
    print("calculate_forecast_accuracy")
```

Figure 16: Forecasted Values