Test Plan and Results

Overall Test Plan

Our testing approach will be heavily focused upon unit testing. We need to ensure that these separate systems are able to work concurrently to help our program function properly. We will test positive and negative scenarios using generated data, making sure that we specifically handle all edge cases sufficiently. We will also implement unit tests to ensure API functionality. Finally, all classes will have complete unit test coverage, which are not listed below as they would be repetitive and tedious.

Test Case Descriptions

DA1.1	**Data Acquisitions Test 1**
DA1.2	This test will ensure a connection can be made to the API
DA1.3	This test will make a basic request to the IEX API and receive a known response back. This will ensure a basic connection to the API in order to ensure a strong and continuous connection can be made.
DA1.4	Inputs: A simple request query to the API asking for a small amount of information
DA1.5	Outputs: A known response from the API that can be compared to a previous response
DA1.6	Normal
DA1.7	Blackbox
DA1.8	Performance
DA1.9	Integration

DA2.1	**Data Acquisitions Test 2**
DA2.2	This test will ensure API request count has not exceeded the limit (This is an additional test strongly associated with DA1.1
DA2.3	Although there is no official API cap for monthly usage of API, for non-commercial developers 100 requests/second can be made. We still want to keep track of this amount and be able to test that our secondly cap is not being met. A simple call to

the account.get_usage(quota_typ) will return a number of message quote usages to make sure that a single developer doesn’t go over the API request quota cap.

DA2.4 Inputs: quota type

DA2.5 Outputs: A boolean to tell whether the returned amount is over the secondly quota

DA2.6 Boundary

DA2.7 Blackbox

DA2.8 Performance

DA2.9 Unit

DA3.1 Data Acquisitions Test 3

DA3.2 The test will ensure that all the erroneous data is removed from the data stream.

DA3.3 This is important because we want to make sure that the data has been properly cleaned. So certain assumptions can be made further down the data acquisition pipeline.

DA3.4 Inputs: Various data types

DA3.5 Outputs: The properly cleaned data

DA3.6 Normal

DA3.7 Whitebox

DA3.8 Functional

DA3.9 Unit

DA4.1 Data Acquisitions Test 4

DA4.2 This test will ensure it can take in filtered data from the data cleaning and be able to format it for proper data storage and visualization

DA4.3 This test will have sample inputs with pre-determined outputs and run the sample inputs through for data formatting code. The test will then compare the data formatting outputs against the pre-determined outputs and check for any discrepancies. The test will pass if no differences are found.

DA4.4 Inputs: Data outputted from the data cleaning portion of our server

DA4.5 Outputs: Data formatted for database storage

DA4.6	Normal
DA4.7	Blackbox
DA4.8	Functional
DA4.9	Unit

DA5.1	**Data Acquisitions Test 5**
DA5.2	This test will send data to the cloud.
DA5.3	This test will ensure that data can be properly sent to the cloud without corruption.
DA5.4	Inputs: Data to be stored on the cloud
DA5.5	Outputs: Success or failure of the operation
DA5.6	Normal
DA5.7	Blackbox
DA5.8	Functional
DA5.9	Unit

DV1.1	**Data Visualization Test 1**
DV1.2	Test for user Queries
DV1.3	This test will verify that a user query will return the correct data formatted for the user in a readable manner.
DV1.4	Input: A user query with multiple different commands for data
DV1.5	Output: A correct result of said query will be returned to the user
DV1.6	Normal
DV1.7	Whitebox
DV1.8	Functional
DV1.9	Integration

DV2.1	**Data Visualization Test 2**

DV2.2	The purpose of this test is to make sure our webapp can display financial data
DV2.3	This test will ensure that data from our data storage can be displayed properly on our web app
DV2.4	Input: A user query
DV2.5	Outputs: A webapp which displays the user data
DV2.6	Normal
DV2.7	Whitebox
DV2.8	Functional
DV2.9	Integration
DV3.1	**Data Visualization Test 3**
DV3.2	This test will ensure multiple different stocks can have their trends displayed at a time, allowing the end user to compare their selected stocks at a glance
DV3.3	This test will use pre-made queries and call for at least two stock trends to be displayed on a graph at the same time. This graph should include a legend, a scale, and one trend line for each stock displayed.
DV3.4	Input:User queries
DV3.5	Output:Graph with data plotted
DV3.6	Normal
DV3.7	Blackbox
DV3.8	Performance
DV3.9	Integration
DV4.1	**Data Visualization Test 4**
DV4.2	This test will ensure that we can retrieve a high number of records from the database
DV4.3	This test is to ensure that we will not have a bottleneck from our data storage
DV4.4	Inputs: A user query
DV4.5	Outputs: Records from the database which will be turned into a database object

DV4.6 Normal

DV4.7 Blackbox

DV4.8 Performance

DV4.9 Integration

MV1.1 Model Validation

MV1.2 This test will ensure that our model integrates into our pipeline

MV1.3 As model parameters grow larger and larger we need to ensure that this does not become too much of a bottle neck.This test is needed in order to ensure that our model is up to our performance standards.

MV1.4 Inputs: The model as well as some input data to the model

MV1.5 Output: The trend data which will be passed into the data visualization

MV1.6 Normal

MV1.7 Blackbox

MV1.8 Performance

MV1.9 Unit

Test Case Matrix

	Normal/ Abnormal/ Boundary	**Blackbox/ Whitebox**	**Functional/ Performance**	**Unit/ Integration**
DA1	Normal	Blackbox	Performance	Integration
DA2	Boundary	Blackbox	Performance	Unit
DA3	Normal	Whitebox	Functional	Unit
DA4	Normal	Blackbox	Functional	Unit
DA5	Normal	Blackbox	Functional	Unit
DV1	Normal	Whitebox	Functional	Integration
DV2	Normal	Whitebox	Functional	Integration
DV3	Normal	Blackbox	Performance	Integration
DV4	Normal	Blackbox	Performance	Integration
MV1	Normal	Blackbox	Performance	Unit