ISM6208 Data Warehousing Final Project
James Long
USF Summer 2021

Contents

Executive Summary

This paper presents the processes for and results of building a very basic predictive model for major U.S. stock indexes based on core economic data. The model is dimensional and based on a relational star schema. I am particularly interested in the influence the Federal Reserve has on the markets, so my model uses predictors aligned with the Fed's mandate (employment and inflation) along with the Federal Funds rate.

This basic model could evolve to include something like a "Fed Sentiment" fact table. The purpose would be to capture and quantify how hawkish or dovish Fed comments are whenever a member of the Fed makes public comments. Natural Language Processing (NLP) or other Artificial Intelligence (AI) techniques could be used to assign a numeric value to the sentiment indicator for each Fed comment. Due to lack of time and data, I modeled this concept but did not implement it.

This project is the culmination of effort performed over the span of a six-week semester. A large set of financial data was already available within the course DBMS, so I relied on it as my primary data source. This enabled me to focus more on modeling, visualizing, and mining. The project incorporates some of the work assigned earlier in the semester and builds upon it to meet the project requirements. Though given the option to work in a group, I worked alone to maximize my learning outcomes.

Problem Statement

Accurately predicting stock moves is notoriously difficult yet extremely alluring due to the potential rewards. Everyone that has any investment in stock (401k, IRA, pension, retail broker, etc.) has a motive to solve this problem. Unfortunately, countless variables influence stock market movements, and the influence of each variable changes daily or even hourly. This creates both opportunity and risk every trading day in every stock market around the globe. For many families, solving this puzzle with even modest reliability could mean the difference between private transportation or public transit, buying a home or renting an apartment, private or public school. Otherwise, long term investing is the only safe option. While that helps with retirement, it does not pay the bills due today. Data mining and visualization could uncover patterns that help to solve this challenge.

Data Collection and Preparation

Most of the data I used were already in the FIN schema. I replicated these data to my personal schema and modified them as needed by dropping columns, adding columns, renaming columns, populating columns, etc. In the case of the Employment_Facts table, I also dropped some rows of older data that preceded the earliest month in the Cal_Month_Dim_MV materialized view.

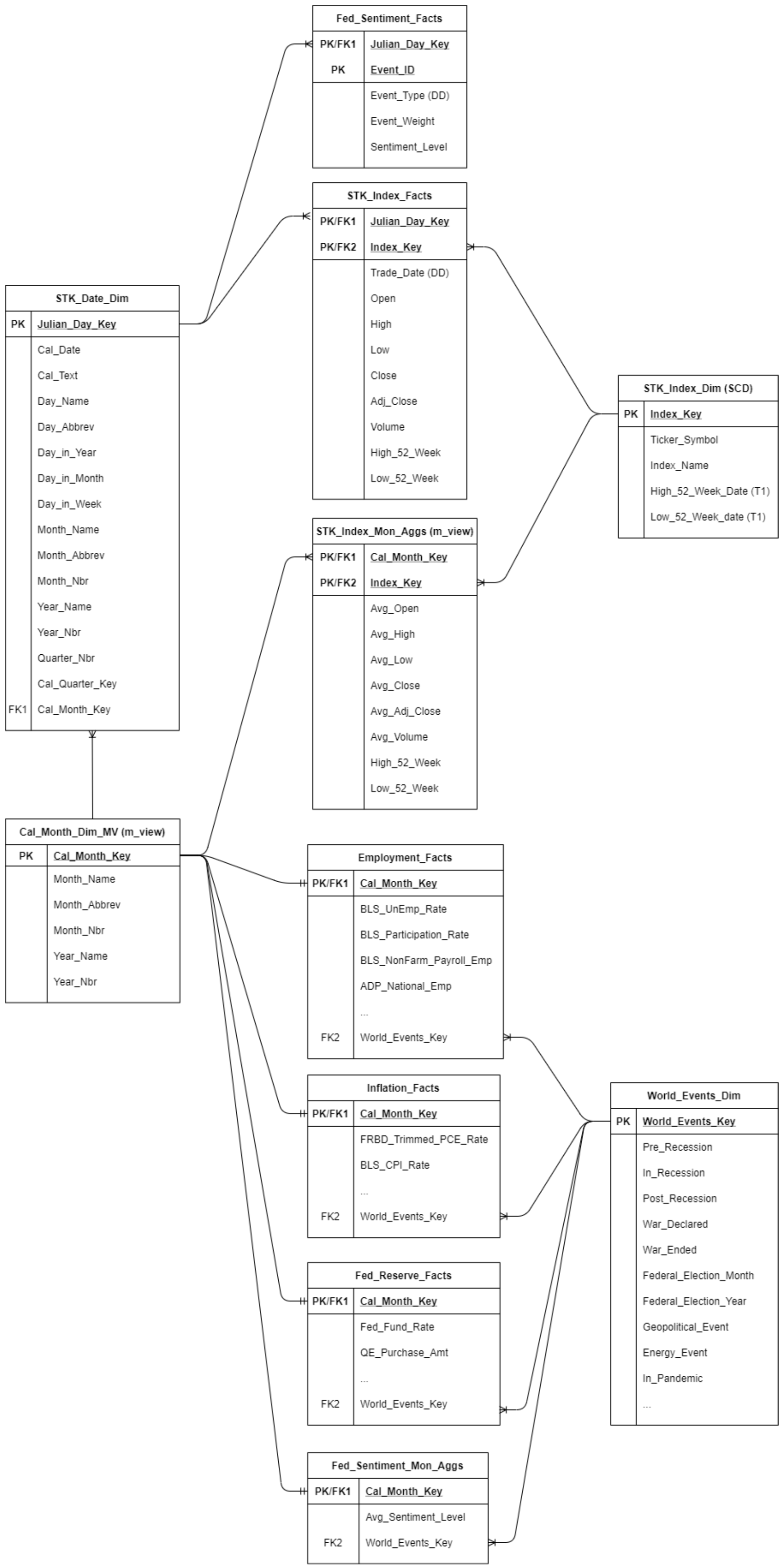
No data were available in the DMBS for inflation rates. So, I searched the FRED site for relevant series. Many series were indexed to a particular year, so they would not correlate with other data that fluctuated monthly. Fortunately, I found a mean Consumer Price Index (CPI) series produced by the Federal Reserve Bank of Dallas (FRBD) on the monthly basis. Prior to loading that into my schema, I dropped the rows of newer data that succeeded the latest month in the Cal_Month_Dim_MV materialized view (July 2018).

No data were available in the DMBS for Fed fund rates after March 2012. So, I downloaded the same series from the FRED site, dropped the rows that were already in the DBMS, and then inserted new rows into the Fed_Reserve_Facts table to fill in missing data from April 2012 to July 2018.

No data were available in the DMBS for BLS unemployment rates after March 2015. So, I downloaded the same series from the FRED site, dropped the rows that were already in the DBMS, and then inserted new rows into the Employments_Facts table to fill in missing data from April 2015 to July 2018.

Database Design

The following ERD depicts the dimensional schema I created for this project.


Fed_Sentiment_Facts
PK/FK1 Julian_Day_Key
PK Event_ID
Event_Type (DD)
Event_Weight
Sentiment_Level
STK_Index_Facts
PK/FK1 Julian_Day_Key
PK/FK2 Index_Key
Trade_Date (DD)
Open
High
Low
Close
Adj_Close
Volume
High_52_Week
Low_52_Week
STK_Date_Dim
PK Julian_Day_Key
Cal_Date
Cal_Text
Day_Name
Day_Abbrev
Day_in_Year
Day_in_Month
Day_in_Week
Month_Name
Month_Abbrev
Month_Nbr
Year_Name
Year_Nbr
Quarter_Nbr
Cal_Quarter_Key
FK1 Cal_Month_Key
STK_Index_Dim (SCD)
PK Index_Key
Ticker_Symbol
Index_Name
High_52_Week_Date (T1)
Low_52_Week_date (T1)
STK_Index_Mon_Aggs (m_view)
PK/FK1 Cal_Month_Key
PK/FK2 Index_Key
Avg_Open
Avg_High
Avg_Low
Avg_Close
Avg_Adj_Close
Avg_Volume
High_52_Week
Low_52_Week
Cal_Month_Dim_MV (m_view)
PK Cal_Month_Key
Month_Name
Month_Abbrev
Month_Nbr
Year_Name
Year_Nbr
Employment_Facts
PK/FK1 Cal_Month_Key
BLS_UnEmp_Rate
BLS_Participation_Rate
BLS_NonFarm_Payroll_Emp
ADP_National_Emp
...
FK2 World_Events_Key
Inflation_Facts
PK/FK1 Cal_Month_Key
FRBD_Trimmed_PCE_Rate
BLS_CPI_Rate
...
FK2 World_Events_Key
World_Events_Dim
PK World_Events_Key
Pre_Recession
In_Recession
Post_Recession
War_Declared
War_Ended
Federal_Election_Month
Federal_Election_Year
Geopolitical_Event
Energy_Event
In_Pandemic
...
Fed_Reserve_Facts
PK/FK1 Cal_Month_Key
Fed_Fund_Rate
QE_Purchase_Amt
...
FK2 World_Events_Key
Fed_Sentiment_Mon_Aggs
PK/FK1 Cal_Month_Key
Avg_Sentiment_Level
FK2 World_Events_Key

Monthly Grain

I am not interested in quarterly aggregates because I want to know how to trade index ETFs leading up to Fed meetings, public Fed comments, events featuring Fed speakers, etc. Too many such events occur in a quarter for that time span to be useful. So, I only created a monthly aggregate for the date table. Most of the fact tables contain data produced on a monthly basis, so this grain works well for aggregation.

52 Week Data

The “52 week” facts in the STK_Index_Mon_Aggs materialized view record the highest and lowest values during each aggregate period (not the average). The “52 week” dates in the STK_Index_Dim table are based on the trailing twelve months.

World Events

The World_Events_Dim table attempts to capture some of the market's psychological factors. This could help explain temporary deviations from the normal correlation of indicator facts to index price movements. For example, Pre_Recession would indicate the month prior to the formal start of a recession, whereas Post_Recession would indicate the month after the formal end of a recession. Some of the other attributes probably should be modeled granularly like recession. For now, they serve as placeholders to indicate the types of events that might be relevant.

Fed Sentiment

The grain of the Fed_Sentiment_Facts table is daily, so we expect the fact table to be sparsely populated. The date alone is insufficient to uniquely identify each row because multiple events can occur on a single day. So, a system-generated identity field is included in the primary key. An event type is included to inform us of the nature of the event at which comments were made, but this is modeled as a degenerate dimension on the assumption we do not need to know anything about specific events. An event weight is assigned to each event type to indicate the relevant importance. This enables us to calculate a weighted average of the sentiment level in the monthly aggregate view. We need this because some comments (e.g., FOMC meeting minutes or the dot plot) have more impact on market psychology than other comments (e.g., a Fed President speaking at a business conference). This sentiment concept was modeled but not instantiated since I had no data.

SQL Code

I issued the following SQL commands to instantiate the tables within my personal schema (DW787).

```
CREATE TABLE
  STK_DATE_DIM
AS
  SELECT
    *
  FROM
    FIN.STK_DATE_DIMS SDD
  INNER JOIN
```

```
    (SELECT
      JULIAN_DAY_KEY AS DAY_KEY,
      TO_NUMBER(TO_CHAR(CAL_DATE, 'YYYYMM'),'999999') AS CAL_MONTH_KEY
    FROM
      FIN.STK_DATE_DIMS) CMK
  ON SDD.JULIAN_DAY_KEY=CMK.DAY_KEY;

ALTER TABLE
  STK_DATE_DIM
DROP COLUMN DAY_KEY;

ALTER TABLE
  STK_DATE_DIM
ADD CONSTRAINT
  STK_DATE_DIM_PK
PRIMARY KEY
  (JULIAN_DAY_KEY);

ALTER TABLE
  STK_DATE_DIM
ADD CONSTRAINT
  STK_DATE_MONTH_FK
FOREIGN KEY
  (CAL_MONTH_KEY)
REFERENCES
  CAL_MONTH_DIM_MV(CAL_MONTH_KEY);

CREATE TABLE
  STK_INDEX_DIM
AS
  SELECT
    *
  FROM
    FIN.STK_INDEX_DIMS;

ALTER TABLE
  STK_INDEX_DIM
ADD CONSTRAINT
  STK_INDEX_DIM_PK
PRIMARY KEY
  (INDEX_KEY);
```

```
ALTER TABLE
   STK_INDEX_DIM
ADD HIGH_52_WEEK_DATE DATE;

ALTER TABLE
   STK_INDEX_DIM
ADD LOW_52_WEEK_DATE DATE;

CREATE TABLE
   STK_INDEX_FACTS
AS
   SELECT
      *
   FROM
      FIN.STK_INDEX_FACTS IF
      INNER JOIN
         (SELECT
            TRADE_DATE AS TD,
            INDEX_KEY AS IK,
            ROUND(MAX(HIGH) OVER (PARTITION BY INDEX_KEY ORDER BY TRADE_DATE ASC RANGE
INTERVAL '365' DAY(3) PRECEDING),2) AS HIGH_52_WEEK,
            ROUND(MIN(LOW) OVER (PARTITION BY INDEX_KEY ORDER BY TRADE_DATE ASC RANGE
INTERVAL '365' DAY(3) PRECEDING),2) AS LOW_52_WEEK
         FROM
            FIN.STK_INDEX_FACTS
         ORDER BY TRADE_DATE, INDEX_KEY) HL
         ON IF.TRADE_DATE=HL.TD AND IF.INDEX_KEY=HL.IK;

ALTER TABLE
   STK_INDEX_FACTS
DROP COLUMN TD;

ALTER TABLE
   STK_INDEX_FACTS
DROP COLUMN IK;

ALTER TABLE
   STK_INDEX_FACTS
ADD CONSTRAINT
   STK_INDEX_FACTS_PK
PRIMARY KEY
   (INDEX_KEY, JULIAN_DAY_KEY);

ALTER TABLE
   STK_INDEX_FACTS
ADD CONSTRAINT
   STK_INDEX_INDEX_FK
FOREIGN KEY
```

```
   (INDEX_KEY)
REFERENCES
   STK_INDEX_DIM(INDEX_KEY);

ALTER TABLE
   STK_INDEX_FACTS
ADD CONSTRAINT
   STK_INDEX_JULIAN_FK
FOREIGN KEY
   (JULIAN_DAY_KEY)
REFERENCES
   STK_DATE_DIM(JULIAN_DAY_KEY);

CREATE MATERIALIZED VIEW
   CAL_MONTH_DIM_MV
BUILD IMMEDIATE
REFRESH FORCE
ON DEMAND
AS
   SELECT DISTINCT
     CAL_MONTH_KEY,
     MONTH_NAME,
     MONTH_ABBREV,
     MONTH_NBR,
     YEAR_NAME,
     YEAR_NBR
   FROM
     STK_DATE_DIM;

ALTER MATERIALIZED VIEW
   CAL_MONTH_DIM_MV
ADD CONSTRAINT
   CAL_MONTH_DIM_MV_PK PRIMARY KEY("CAL_MONTH_KEY");

ALTER MATERIALIZED VIEW
   CAL_MONTH_DIM_MV
ADD CONSTRAINT
   CAL_MONTH_DIM_MV_PK_NOT_NULL check("CAL_MONTH_KEY" IS NOT NULL) ENABLE;

EXEC DBMS_STATS.GATHER_TABLE_STATS('DW787', 'CAL_MONTH_DIM_MV');

CREATE MATERIALIZED VIEW
   STK_INDEX_MON_AGGS
BUILD IMMEDIATE
REFRESH FORCE
ON DEMAND
AS
   SELECT
```

```
        CAL_MONTH_KEY,
        INDEX_KEY,
        ROUND(AVG(OPEN),2) AS AVG_OPEN,
        ROUND(AVG(HIGH),2) AS AVG_HIGH,
        ROUND(AVG(LOW),2) AS AVG_LOW,
        ROUND(AVG(CLOSE),2) AS AVG_CLOSE,
        ROUND(AVG(ADJ_CLOSE),2) AS AVG_ADJ_CLOSE,
        ROUND(AVG(VOLUME),0) AS AVG_VOLUME,
        MAX(HIGH_52_WEEK) AS HIGH_52_WEEK,
        MIN(LOW_52_WEEK) AS LOW_52_WEEK
    FROM
        STK_DATE_DIM DD
        INNER JOIN STK_INDEX_FACTS IF
            ON DD.JULIAN_DAY_KEY=IF.JULIAN_DAY_KEY
    GROUP BY CAL_MONTH_KEY, INDEX_KEY
    ORDER BY CAL_MONTH_KEY, INDEX_KEY;

ALTER MATERIALIZED VIEW
    STK_INDEX_MON_AGGS
ADD CONSTRAINT
    SYS_C12345678 check("CAL_MONTH_KEY" IS NOT NULL) ENABLE;

ALTER MATERIALIZED VIEW
    STK_INDEX_MON_AGGS
ADD CONSTRAINT
    STK_INDEX_MON_AGGS_PK PRIMARY KEY("CAL_MONTH_KEY","INDEX_KEY");

ALTER MATERIALIZED VIEW
    STK_INDEX_MON_AGGS
ADD CONSTRAINT
    STK_INDEX_AGGS_CAL_MON_FK
FOREIGN KEY
    (CAL_MONTH_KEY)
REFERENCES
    CAL_MONTH_DIM_MV(CAL_MONTH_KEY);

ALTER MATERIALIZED VIEW
    STK_INDEX_MON_AGGS
ADD CONSTRAINT
    STK_INDEX_AGGS_INDEX_FK
FOREIGN KEY
    (INDEX_KEY)
REFERENCES
    STK_INDEX_DIM(INDEX_KEY);

EXEC DBMS_STATS.GATHER_TABLE_STATS('DW787', 'STK_INDEX_MON_AGGS');

UPDATE
```

```
  STK_INDEX_DIM
SET
  HIGH_52_WEEK_DATE=
          (SELECT
            MIN(TRADE_DATE)
          FROM
            STK_INDEX_FACTS
          WHERE
            INDEX_KEY=101
            AND TRADE_DATE >='11-JUL-17'
            AND HIGH_52_WEEK=(SELECT
                      MAX(HIGH_52_WEEK)
                    FROM
                      STK_INDEX_FACTS
                    WHERE
                      INDEX_KEY=101
                      AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=101;

UPDATE
  STK_INDEX_DIM
SET
  HIGH_52_WEEK_DATE=
          (SELECT
            MIN(TRADE_DATE)
          FROM
            STK_INDEX_FACTS
          WHERE
            INDEX_KEY=102
            AND TRADE_DATE >='11-JUL-17'
            AND HIGH_52_WEEK=(SELECT
                      MAX(HIGH_52_WEEK)
                    FROM
                      STK_INDEX_FACTS
                    WHERE
                      INDEX_KEY=102
                      AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=102;

UPDATE
  STK_INDEX_DIM
SET
  HIGH_52_WEEK_DATE=
          (SELECT
            MIN(TRADE_DATE)
          FROM
```

```
          STK_INDEX_FACTS
        WHERE
          INDEX_KEY=103
          AND TRADE_DATE >='11-JUL-17'
          AND HIGH_52_WEEK=(SELECT
                    MAX(HIGH_52_WEEK)
                  FROM
                    STK_INDEX_FACTS
                  WHERE
                    INDEX_KEY=103
                    AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=103;

UPDATE
  STK_INDEX_DIM
SET
  LOW_52_WEEK_DATE=
        (SELECT
          MIN(TRADE_DATE)
        FROM
          STK_INDEX_FACTS
        WHERE
          INDEX_KEY=101
          AND TRADE_DATE >='11-JUL-17'
          AND LOW_52_WEEK=(SELECT
                    MIN(LOW_52_WEEK)
                  FROM
                    STK_INDEX_FACTS
                  WHERE
                    INDEX_KEY=101
                    AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=101;

UPDATE
  STK_INDEX_DIM
SET
  LOW_52_WEEK_DATE=
        (SELECT
          MIN(TRADE_DATE)
        FROM
          STK_INDEX_FACTS
        WHERE
          INDEX_KEY=102
          AND TRADE_DATE >='11-JUL-17'
          AND LOW_52_WEEK=(SELECT
                    MIN(LOW_52_WEEK)
```

```
                    FROM
                      STK_INDEX_FACTS
                    WHERE
                      INDEX_KEY=102
                      AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=102;

UPDATE
  STK_INDEX_DIM
SET
  LOW_52_WEEK_DATE=
          (SELECT
            MIN(TRADE_DATE)
          FROM
            STK_INDEX_FACTS
          WHERE
            INDEX_KEY=103
            AND TRADE_DATE >='11-JUL-17'
            AND LOW_52_WEEK=(SELECT
                      MIN(LOW_52_WEEK)
                    FROM
                      STK_INDEX_FACTS
                    WHERE
                      INDEX_KEY=103
                      AND TRADE_DATE >='11-JUL-17'))
WHERE
  INDEX_KEY=103;

CREATE TABLE WORLD_EVENTS_DIM (
  WORLD_EVENTS_KEY NUMBER(4,0) GENERATED BY DEFAULT AS IDENTITY,
  PRE_RECESSION VARCHAR2(5 BYTE),
  IN_RECESSION VARCHAR2(5 BYTE),
  POST_RECESSION VARCHAR2(5 BYTE),
  WAR_DECLARED VARCHAR2(5 BYTE),
  WAR_ENDED VARCHAR2(5 BYTE),
  FEDERAL_ELECTION_MONTH VARCHAR2(5 BYTE),
  FEDERAL_ELECTION_YEAR VARCHAR2(5 BYTE),
  GEOPOLITICAL_EVENT VARCHAR2(5 BYTE),
  ENERGY_EVENT VARCHAR2(5 BYTE),
  IN_PANDEMIC VARCHAR2(5 BYTE)
);

ALTER TABLE
  WORLD_EVENTS_DIM
ADD CONSTRAINT
  WORLD_EVENTS_DIM_PK PRIMARY KEY ("WORLD_EVENTS_KEY");
```

```
INSERT INTO WORLD_EVENTS_DIM (
   PRE_RECESSION,
   IN_RECESSION,
   POST_RECESSION,
   WAR_DECLARED,
   WAR_ENDED,
   FEDERAL_ELECTION_MONTH,
   FEDERAL_ELECTION_YEAR,
   GEOPOLITICAL_EVENT,
   ENERGY_EVENT,
   IN_PANDEMIC)
VALUES
   (NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

CREATE TABLE
   EMPLOYMENT_FACTS
AS
   SELECT
     CAL_MONTH_KEY,
     UNRATE_PERCENT AS BLS_UNEMP_RATE,
     CAST(NULL AS NUMBER) AS BLS_PARTICIPATION_RATE,
     CAST(NULL AS NUMBER) AS BLS_NONFARM_PAYROLL_EMP,
     CAST(NULL AS NUMBER) AS ADP_NATIONAL_EMP,
     1 AS WORLD_EVENTS_KEY
   FROM
     FIN.FRED_UNRATE;

ALTER TABLE
   EMPLOYMENT_FACTS
ADD CONSTRAINT
   EMPLOYMENT_FACTS_PK PRIMARY KEY ("CAL_MONTH_KEY");

DELETE
FROM
   EMPLOYMENT_FACTS
WHERE
   CAL_MONTH_KEY IN (194801, 194802, 194803, 194804, 194805, 194806, 194807, 194808, 194809,
194810, 194811, 194812, 194901, 194902, 194903, 194904, 194905, 194906, 194907, 194908, 194909,
194910, 194911, 194912);

ALTER TABLE
   EMPLOYMENT_FACTS
ADD CONSTRAINT
   EMPLOYMENT_FACTS_MONTH_FK
FOREIGN KEY
   ("CAL_MONTH_KEY")
REFERENCES
   "CAL_MONTH_DIM_MV"("CAL_MONTH_KEY");
```

```
ALTER TABLE
   EMPLOYMENT_FACTS
ADD CONSTRAINT
   EMPLOYMENT_FACTS_WORLD_FK
FOREIGN KEY
   ("WORLD_EVENTS_KEY")
REFERENCES
   "WORLD_EVENTS_DIM"("WORLD_EVENTS_KEY");

CREATE TABLE
   FED_RESERVE_FACTS
AS
   SELECT
     CAL_MONTH_KEY,
     FF_RATE AS FED_FUND_RATE,
     CAST(NULL AS NUMBER) AS QE_PURCHASE_AMT,
     1 AS WORLD_EVENTS_KEY
   FROM
     FIN.FRED_FEDFUNDS;

ALTER TABLE
   FED_RESERVE_FACTS
ADD CONSTRAINT
   FED_RESERVE_FACTS_PK PRIMARY KEY ("CAL_MONTH_KEY");

ALTER TABLE
   FED_RESERVE_FACTS
ADD CONSTRAINT
   FRF_PK_NOT_NULL CHECK ("CAL_MONTH_KEY" IS NOT NULL) ENABLE;

ALTER TABLE
   FED_RESERVE_FACTS
ADD CONSTRAINT
   FED_RESERVE_MONTH_FK
FOREIGN KEY
   ("CAL_MONTH_KEY")
REFERENCES
   "CAL_MONTH_DIM_MV" ("CAL_MONTH_KEY");

ALTER TABLE
   FED_RESERVE_FACTS
ADD CONSTRAINT
   FED_RESERVE_WORLD_FK
FOREIGN KEY
   ("WORLD_EVENTS_KEY")
REFERENCES
   "WORLD_EVENTS_DIM" ("WORLD_EVENTS_KEY");
```

To load inflation data, I downloaded the FRED data in CSV format and created a temporary table to hold the data.

```
CREATE TABLE INFLATION_TEMP (
  PUB_DATE DATE,
  PCE NUMBER(5,2)
);
```

Next, I used the import wizard in SQL Developer to load the source data into the INFLATION_TEMP table. Then I created the INFLATION_FACTS table.

```
CREATE TABLE
  INFLATION_FACTS
AS
  SELECT
   TO_NUMBER(TO_CHAR(PUB_DATE, 'YYYYMM'),'999999') AS CAL_MONTH_KEY,
   PCE AS FRBD_TRIMMED_PCE_RATE,
   CAST(NULL AS NUMBER) AS BLS_CPI_RATE,
   1 AS WORLD_EVENTS_KEY
  FROM
   INFLATION_TEMP;

DROP TABLE INFLATION_TEMP;

ALTER TABLE
  INFLATION_FACTS
ADD CONSTRAINT
  INFLATION_FACTS_PK PRIMARY KEY ("CAL_MONTH_KEY");

ALTER TABLE
  INFLATION_FACTS
ADD CONSTRAINT
  INFLATION_FACTS_PK_NOT_NULL CHECK ("CAL_MONTH_KEY" IS NOT NULL) ENABLE;

ALTER TABLE
  INFLATION_FACTS
ADD CONSTRAINT
  INFLATION_FACTS_MONTH_FK
FOREIGN KEY
  ("CAL_MONTH_KEY")
REFERENCES
  "CAL_MONTH_DIM_MV" ("CAL_MONTH_KEY");

ALTER TABLE
  INFLATION_FACTS
ADD CONSTRAINT
  INFLATION_FACTS_WORLD_FK
```

```
FOREIGN KEY
  ("WORLD_EVENTS_KEY")
REFERENCES
  "WORLD_EVENTS_DIM" ("WORLD_EVENTS_KEY");
```

To load additional rows of newer Fed funds rate data, I downloaded the FRED data in CSV format and then used Excel functions to extract the year and month from the date field. These were concatenated into a Cal_Month_Key field in the CSV file. I also created a World_Events_Key field in the CSV file with a value of 1 in each row. I then used the import wizard in SQL Developer to load newer data into the Fed_Reserve_Facts table.

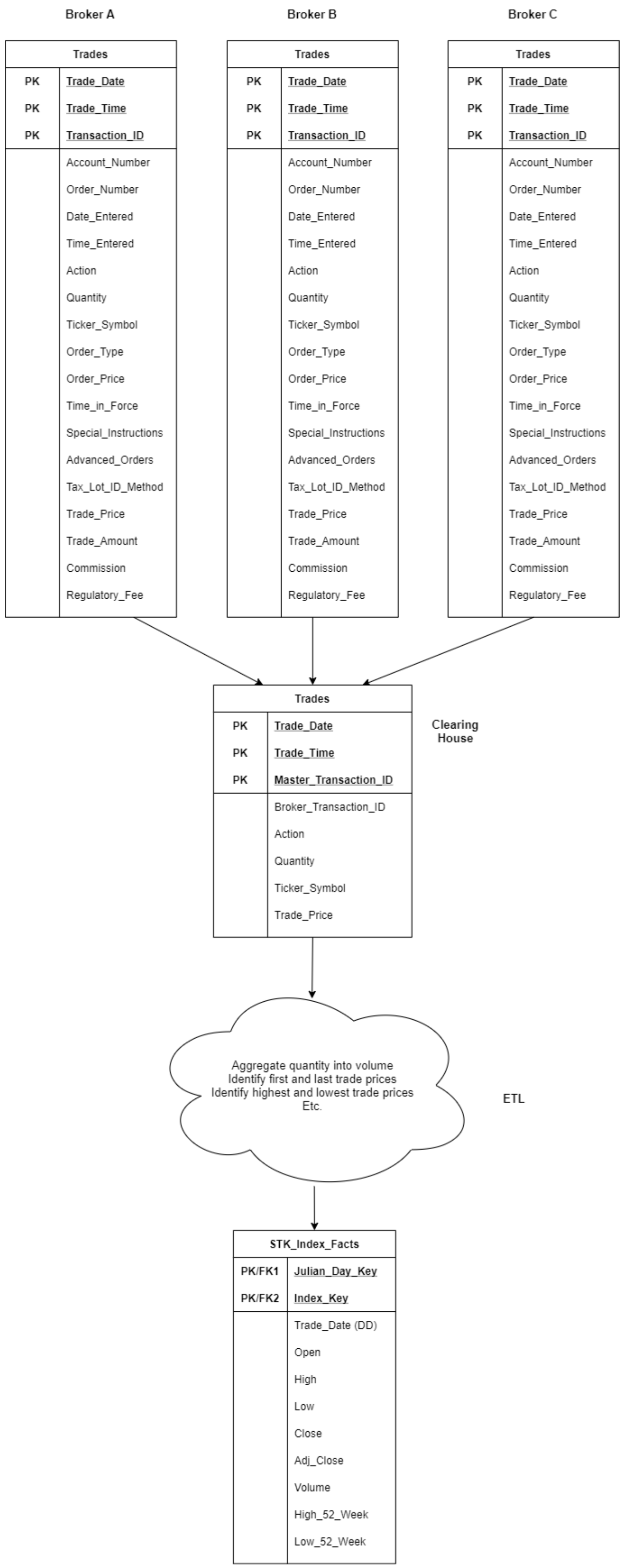
To load additional rows of newer BLS unemployment rate data, I followed the same procedure as for the Fed funds rate data except the data were loaded into the Employment_Facts table.

Following the June 21 lecture, I issued the following SQL statements.

```
ALTER MATERIALIZED VIEW
  CAL_MONTH_DIM_MV ENABLE QUERY REWRITE;

ALTER MATERIALIZED VIEW
  STK_INDEX_MON_AGGS ENABLE QUERY REWRITE;
```

Operational Model

The operational systems that generate trade fact data might look something like the following ERD. This partial model is shown with a high-level view of the ETL process.


Broker A
Broker B
Broker C
Trades
PK Trade_Date
PK Trade_Time
PK Transaction_ID
Account_Number
Order_Number
Date_Entered
Time_Entered
Action
Quantity
Ticker_Symbol
Order_Type
Order_Price
Time_in_Force
Special_Instructions
Advanced_Orders
Tax_Lot_ID_Method
Trade_Price
Trade_Amount
Commission
Regulatory_Fee
Trades
PK Trade_Date
PK Trade_Time
PK Master_Transaction_ID
Broker_Transaction_ID
Action
Quantity
Ticker_Symbol
Trade_Price
Clearing House
Aggregate quantity into volume
Identify first and last trade prices
Identify highest and lowest trade prices
Etc.
ETL
STK_Index_Facts
PK/FK1 Julian_Day_Key
PK/FK2 Index_Key
Trade_Date (DD)
Open
High
Low
Close
Adj_Close
Volume
High_52_Week
Low_52_Week

Reporting

<u>Query 1 - Copied from Assignment 2</u>

This query examines the correlation of the Fed Funds rate with the BLS unemployment rate on a calendar year basis and reports the strongest negative and positive correlations along with the total number of negative and positive correlations.

```
select
  *
from
  (SELECT
    MIN(rate_corr) as "strongest neg corr",
    COUNT(*) as "number of neg corr"
  FROM
    (SELECT
      year_nbr,
      round(corr(fed_fund_rate, bls_unemp_rate),4) as rate_corr
    FROM
      employment_facts ef
      inner join cal_month_dim_mv md
                                on ef.cal_month_key=md.cal_month_key
      inner join fed_reserve_facts frf
        on md.cal_month_key=frf.cal_month_key
    group by year_nbr
    order by rate_corr)
  where rate_corr < 0)
  full join
  (SELECT
    MAX(rate_corr) as "strongest pos correlation",
    COUNT(*) as "number of pos correlation"
  FROM
    (SELECT
      year_nbr,
      round(corr(fed_fund_rate, bls_unemp_rate),4) as rate_corr
    FROM
      employment_facts ef
      inner join cal_month_dim_mv md
        on ef.cal_month_key=md.cal_month_key
      inner join fed_reserve_facts frf
        on md.cal_month_key=frf.cal_month_key
    group by year_nbr
    order by rate_corr)
  where rate_corr > 0)
  on 1=0;
```

<u>Query 2 - Adapted from Assignment 2</u>

This query examines the correlation of stock index price range with volume on an index, monthly, and yearly basis.

```
SELECT
  *
FROM
  (SELECT
    ROUND(CORR(avg_range, avg_volume),4) as Index_Corr
  FROM
    (SELECT
      index_name,
      (CASE
        WHEN GROUPING(year_nbr) = 1 THEN 'Avg'
        ELSE TO_CHAR(year_nbr)
        END) as year,
      (CASE
        WHEN GROUPING(month_nbr) = 1 THEN 'Avg'
        ELSE TO_CHAR(month_nbr)
        END) as month,
      (ROUND(AVG(high),2)-ROUND(AVG(low),2)) as avg_range,
      ROUND(AVG(volume)) as avg_volume
    FROM
      stk_date_dim dd
      inner join stk_index_facts if
        on dd.julian_day_key = if.julian_day_key
      inner join stk_index_dim id
        on if.index_key = id.index_key
    GROUP BY index_name, ROLLUP(year_nbr, month_nbr)
    ORDER BY index_name, year_nbr, month_nbr)
    where year = 'Avg')
  full join
  (SELECT
    ROUND(CORR(avg_range, avg_volume),4) as Yearly_Corr
  FROM
    (SELECT
      index_name,
      (CASE
        WHEN GROUPING(year_nbr) = 1 THEN 'Avg'
        ELSE TO_CHAR(year_nbr)
        END) as year,
      (CASE
        WHEN GROUPING(month_nbr) = 1 THEN 'Avg'
        ELSE TO_CHAR(month_nbr)
        END) as month,
      (ROUND(AVG(high),2)-ROUND(AVG(low),2)) as avg_range,
      ROUND(AVG(volume)) as avg_volume
    FROM
```

```
        stk_date_dim dd
        inner join stk_index_facts if
          on dd.julian_day_key = if.julian_day_key
        inner join stk_index_dim id
          on if.index_key = id.index_key
      GROUP BY index_name, year_nbr, ROLLUP(month_nbr)
      ORDER BY index_name, year_nbr, month_nbr)
      where month = 'Avg'
      and avg_volume != 0)
      on 1=0
    full join
    (SELECT
      ROUND(CORR(avg_range, avg_volume),4) as Monthly_Corr
    FROM
      (SELECT
        index_name,
        year_nbr as year,
        month_nbr as month,
        (ROUND(AVG(high),2)-ROUND(AVG(low),2)) as avg_range,
        ROUND(AVG(volume)) as avg_volume
      FROM
        stk_date_dim dd
        inner join stk_index_facts if
          on dd.julian_day_key = if.julian_day_key
        inner join stk_index_dim id
          on if.index_key = id.index_key
      GROUP BY index_name, year_nbr, month_nbr
      ORDER BY index_name, year_nbr, month_nbr)
      where avg_volume != 0)
      on 1=0;
```

<u>Query 3</u>

This query set reports the three best and worst performing months for each stock index.

```
/* create view to avoid repeatedly generating base data */
CREATE VIEW MONTHLY_AVGS
AS
  SELECT
    index_name,
    (CASE
      WHEN month_nbr = 1 THEN 'Jan'
      WHEN month_nbr = 2 THEN 'Feb'
      WHEN month_nbr = 3 THEN 'Mar'
      WHEN month_nbr = 4 THEN 'Apr'
      WHEN month_nbr = 5 THEN 'May'
      WHEN month_nbr = 6 THEN 'Jun'
      WHEN month_nbr = 7 THEN 'Jul'
```

```
        WHEN month_nbr = 8 THEN 'Aug'
        WHEN month_nbr = 9 THEN 'Sep'
        WHEN month_nbr = 10 THEN 'Oct'
        WHEN month_nbr = 11 THEN 'Nov'
        WHEN month_nbr = 12 THEN 'Dec'
        END) AS MONTH_NAME,
      avg_high,
      avg_low
    FROM
      (SELECT
        index_name,
        (CASE
          WHEN GROUPING(year_nbr) = 1 THEN 'Avg'
          ELSE TO_CHAR(year_nbr)
          END) as year,
        month_nbr,
        ROUND(AVG(high),2) as avg_high,
        ROUND(AVG(low),2) as avg_low
      FROM
        stk_date_dim dd
        inner join stk_index_facts if
          on dd.julian_day_key = if.julian_day_key
        inner join stk_index_dim id
          on if.index_key = id.index_key
      GROUP BY index_name, ROLLUP(year_nbr), month_nbr
      ORDER BY index_name, year_nbr, month_nbr)
    where year = 'Avg'
    ORDER BY INDEX_NAME;

/* best performing months */
SELECT
    *
FROM
    ((SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_high desc) as high_rank
    FROM
      MONTHLY_AVGS
    WHERE index_name = 'Dow Jones Industrial Average'
    ORDER BY HIGH_RANK
    FETCH FIRST 3 ROWS ONLY)
    UNION
    (SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_high desc) as high_rank
    FROM
```

```
      MONTHLY_AVGS
    WHERE index_name = 'S&' || 'P 500'
    ORDER BY HIGH_RANK
    FETCH FIRST 3 ROWS ONLY)
    UNION
    (SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_high desc) as high_rank
    FROM
      MONTHLY_AVGS
    WHERE index_name = 'Russell 2000'
    ORDER BY HIGH_RANK
    FETCH FIRST 3 ROWS ONLY))
ORDER BY HIGH_RANK, INDEX_NAME;

/* worst performing months */
SELECT
    *
FROM
    ((SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_low asc) as low_rank
    FROM
      MONTHLY_AVGS
    WHERE index_name = 'Dow Jones Industrial Average'
    ORDER BY LOW_RANK
    FETCH FIRST 3 ROWS ONLY)
    UNION
    (SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_low asc) as low_rank
    FROM
      MONTHLY_AVGS
    WHERE index_name = 'S&' || 'P 500'
    ORDER BY LOW_RANK
    FETCH FIRST 3 ROWS ONLY)
    UNION
    (SELECT
      index_name,
      MONTH_NAME,
      RANK() OVER (PARTITION BY index_name ORDER BY avg_low asc) as low_rank
    FROM
      MONTHLY_AVGS
    WHERE index_name = 'Russell 2000'
    ORDER BY LOW_RANK
```

```
   FETCH FIRST 3 ROWS ONLY))
ORDER BY LOW_RANK, INDEX_NAME;
```

Modeling

For the modeling requirement, I wanted to use the Oracle Data Miner feature. Since I do not have access to it outside of class, this was a good opportunity to gain exposure. Unfortunately, the student data mining accounts did not have the needed permissions to load data into the data mining tablespace. So, I used Orange instead.

First, I created a flattened materialized view of the intermediate calculations I needed. The earliest record date was inconsistent across the fact tables, so I trimmed them to the oldest common date. The resulting view had 40 years of data, which is still plenty for my analysis. In fact, one could argue that the financial markets are sufficiently different today versus 40 years ago to make data that old useless for a predictive model. Nonetheless, I kept all 40 years. I also chose to focus on the S&P 500 index rather than incorporate all of the indexes.

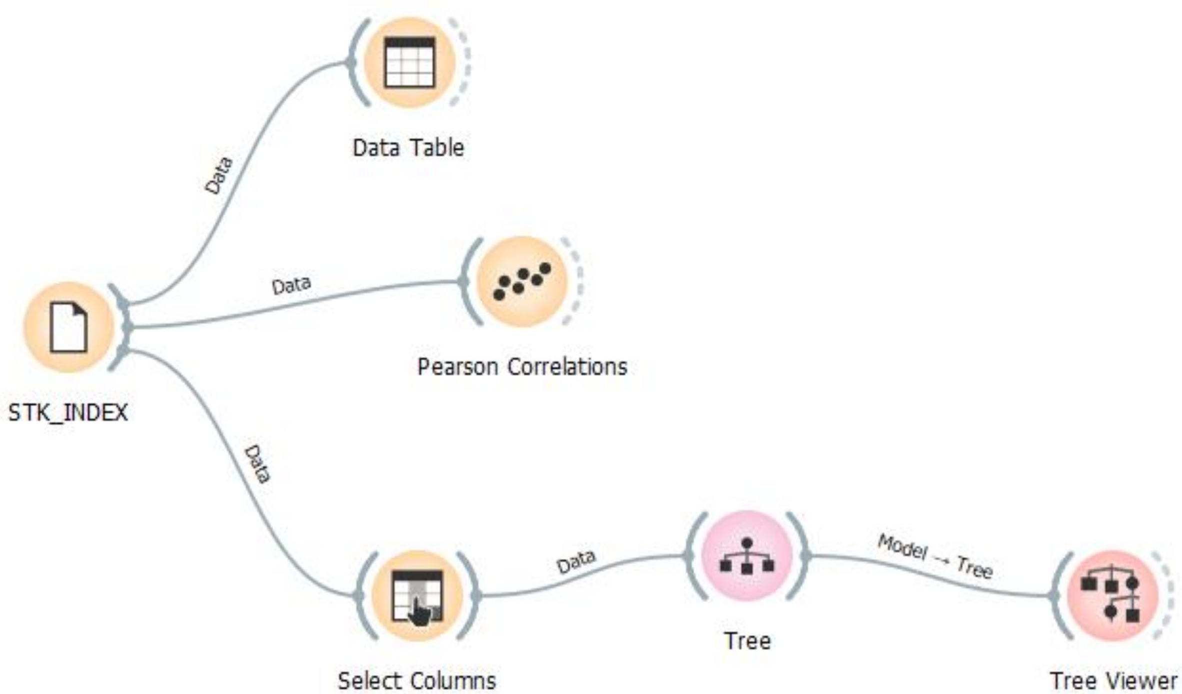
Next, based on the first view, I created another flattened materialized view of the final calculations I needed. Perhaps this could have been accomplished with a single view, but I when I tried, I got SQL windowing errors that I did not have time to resolve. Besides, I think the code is easier to understand with two views, which makes maintenance less error prone. Also, the first view can be used by other derivative views, thus forming the basis of a modular schema to support future data mining efforts.

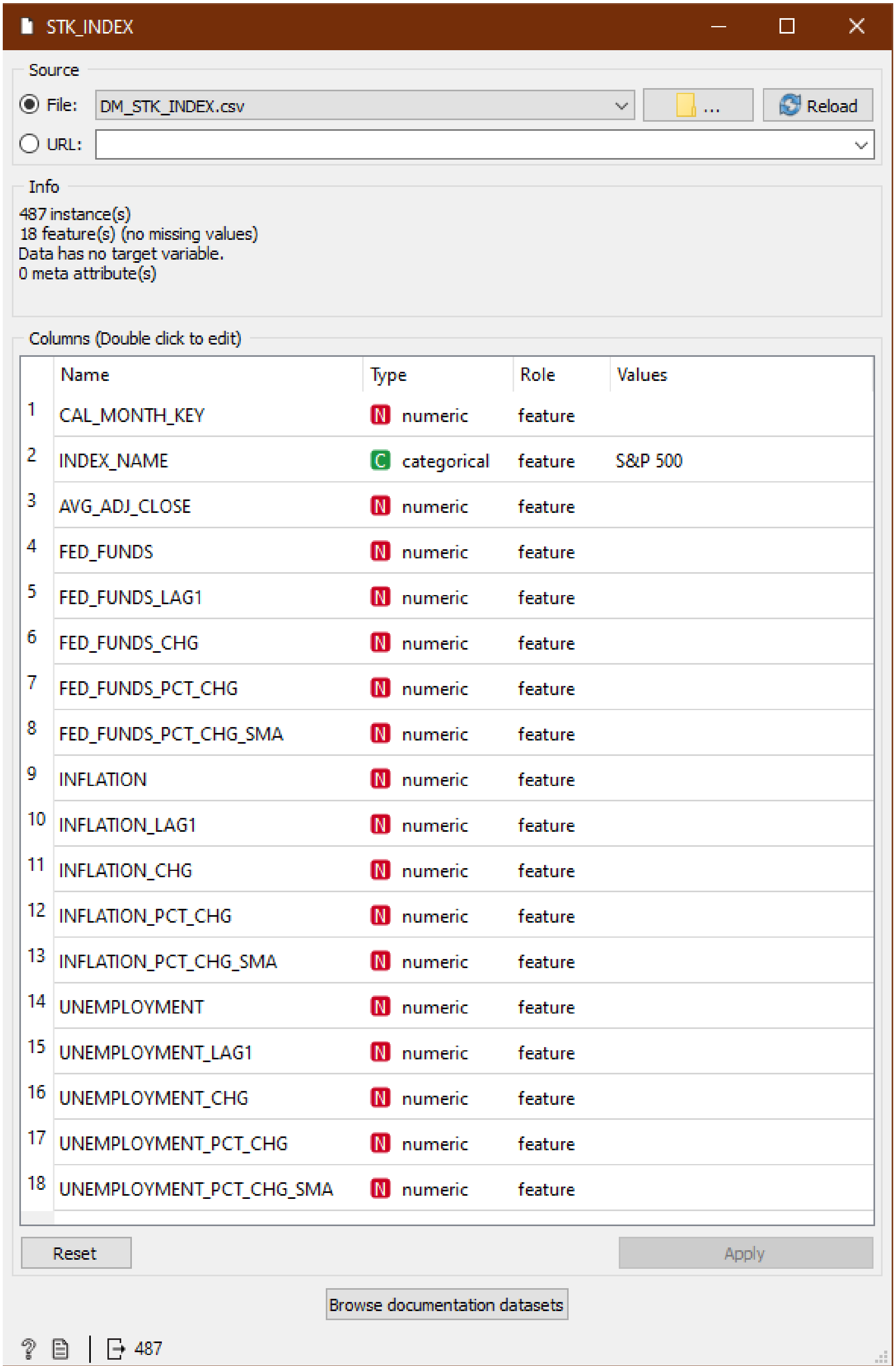
```
CREATE MATERIALIZED VIEW
   DM_INTERMED_MV
AS
   SELECT
      IMA.CAL_MONTH_KEY,
      INDEX_NAME,
      AVG_ADJ_CLOSE,
      FED_FUND_RATE AS FED_FUNDS,
      LAG(FED_FUND_RATE, 1, FED_FUND_RATE)
         OVER (ORDER BY IMA.CAL_MONTH_KEY ASC)
         AS FED_FUNDS_LAG1,
      FED_FUND_RATE - (LAG(FED_FUND_RATE, 1, FED_FUND_RATE)
         OVER (ORDER BY IMA.CAL_MONTH_KEY ASC))
         AS FED_FUNDS_CHG,
      FRBD_TRIMMED_PCE_RATE AS INFLATION,
      LAG(FRBD_TRIMMED_PCE_RATE, 1, FRBD_TRIMMED_PCE_RATE)
         OVER (ORDER BY IMA.CAL_MONTH_KEY ASC)
         AS INFLATION_LAG1,
      FRBD_TRIMMED_PCE_RATE - (LAG(FRBD_TRIMMED_PCE_RATE, 1, FRBD_TRIMMED_PCE_RATE)
         OVER (ORDER BY IMA.CAL_MONTH_KEY ASC))
         AS INFLATION_CHG,
      BLS_UNEMP_RATE AS UNEMPLOYMENT,
      LAG(BLS_UNEMP_RATE, 1, BLS_UNEMP_RATE)
```

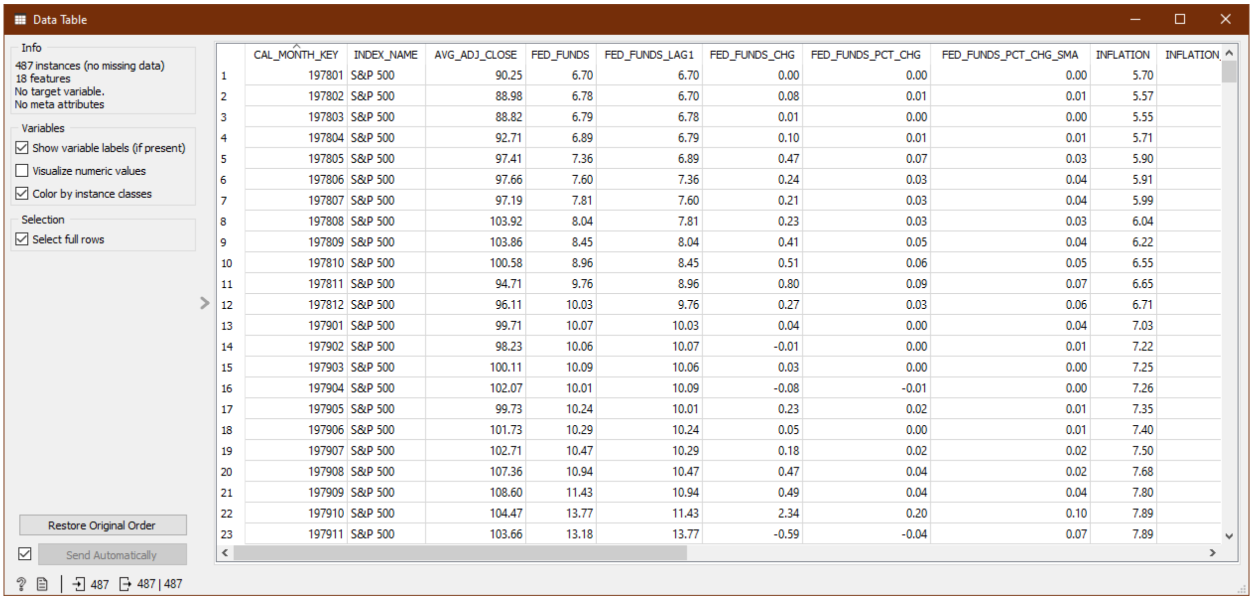
```
      OVER (ORDER BY IMA.CAL_MONTH_KEY ASC)
      AS UNEMPLOYMENT_LAG1,
    BLS_UNEMP_RATE - (LAG(BLS_UNEMP_RATE, 1, BLS_UNEMP_RATE)
      OVER (ORDER BY IMA.CAL_MONTH_KEY ASC))
      AS UNEMPLOYMENT_CHG
  FROM
    STK_INDEX_MON_AGGS IMA
    INNER JOIN FED_RESERVE_FACTS FRF
    ON IMA.CAL_MONTH_KEY=FRF.CAL_MONTH_KEY
    INNER JOIN INFLATION_FACTS IF
    ON IMA.CAL_MONTH_KEY=IF.CAL_MONTH_KEY
    INNER JOIN EMPLOYMENT_FACTS EF
    ON IMA.CAL_MONTH_KEY=EF.CAL_MONTH_KEY
    INNER JOIN STK_INDEX_DIM ID
    ON IMA.INDEX_KEY=ID.INDEX_KEY
  WHERE
    INDEX_NAME='S&'||'P 500'
    AND IMA.CAL_MONTH_KEY >= 197801;

EXEC DBMS_STATS.GATHER_TABLE_STATS('DW787', 'DM_INTERMED_MV');

CREATE MATERIALIZED VIEW
  DM_STK_INDEX_MV
AS
SELECT
  CAL_MONTH_KEY,
  INDEX_NAME,
  AVG_ADJ_CLOSE,
  FED_FUNDS,
  FED_FUNDS_LAG1,
  FED_FUNDS_CHG,
  ROUND(FED_FUNDS_CHG / FED_FUNDS_LAG1,2) AS FED_FUNDS_PCT_CHG,
  ROUND(AVG(FED_FUNDS_CHG / FED_FUNDS_LAG1)
    OVER (ORDER BY CAL_MONTH_KEY ASC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2)
    AS FED_FUNDS_PCT_CHG_SMA,
  INFLATION,
  INFLATION_LAG1,
  INFLATION_CHG,
  ROUND(INFLATION_CHG / INFLATION_LAG1,2) AS INFLATION_PCT_CHG,
  ROUND(AVG(INFLATION_CHG / INFLATION_LAG1)
    OVER (ORDER BY CAL_MONTH_KEY ASC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2)
    AS INFLATION_PCT_CHG_SMA,
  UNEMPLOYMENT,
  UNEMPLOYMENT_LAG1,
  UNEMPLOYMENT_CHG,
  ROUND(UNEMPLOYMENT_CHG / UNEMPLOYMENT_LAG1,2) AS UNEMPLOYMENT_PCT_CHG,
  ROUND(AVG(UNEMPLOYMENT_CHG / UNEMPLOYMENT_LAG1)
    OVER (ORDER BY CAL_MONTH_KEY ASC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2)
```

```
      AS UNEMPLOYMENT_PCT_CHG_SMA
FROM
   DM_INTERMED_MV;

EXEC DBMS_STATS.GATHER_TABLE_STATS('DW787', 'DM_STK_INDEX_MV');
```

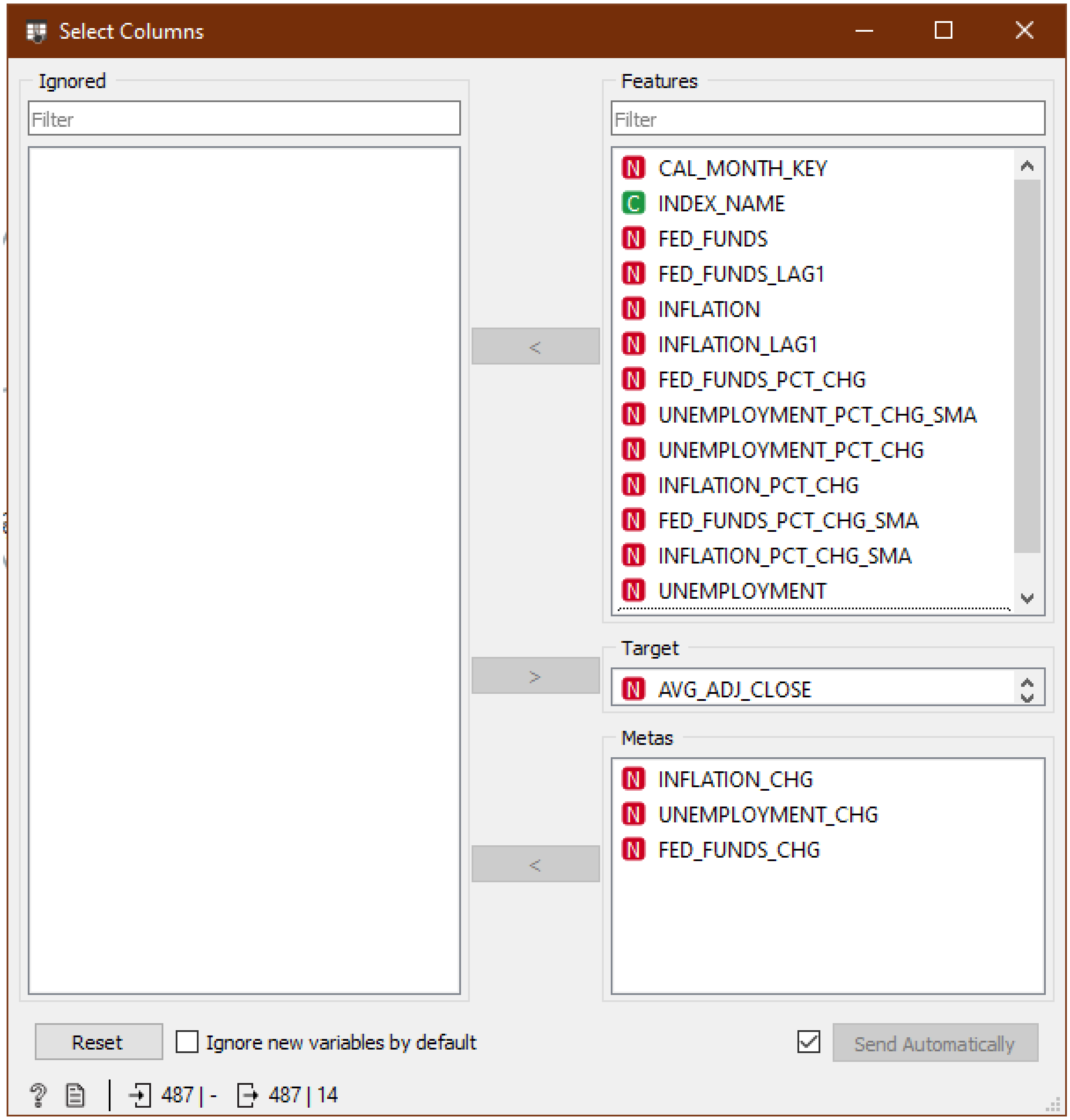
I exported the second view to a CSV file and imported into Orange. Then I followed the same decision tree analysis that was demonstrated in class. My target was AVG_ADJ_CLOSE, and my initial meta was FED_FUNDS_PCT_CHG_SMA. However, the resulting tree had 347 nodes, 174 leaves, and was 14 levels deep. So, I tried various other metas alone and in combinations. The results were all similar.

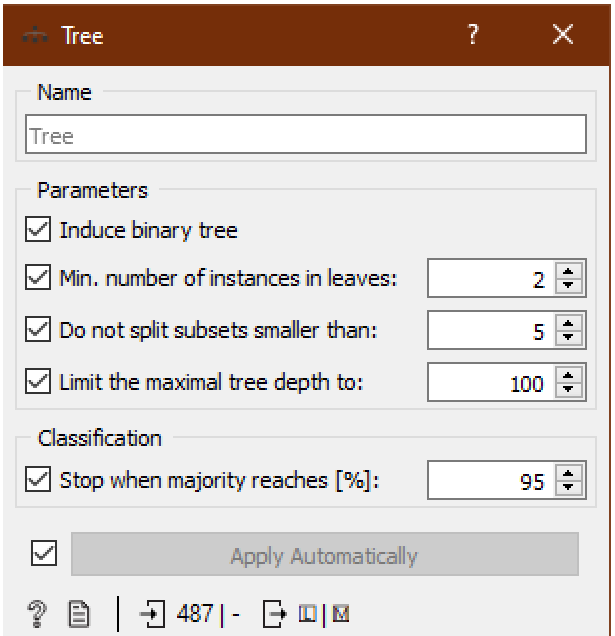


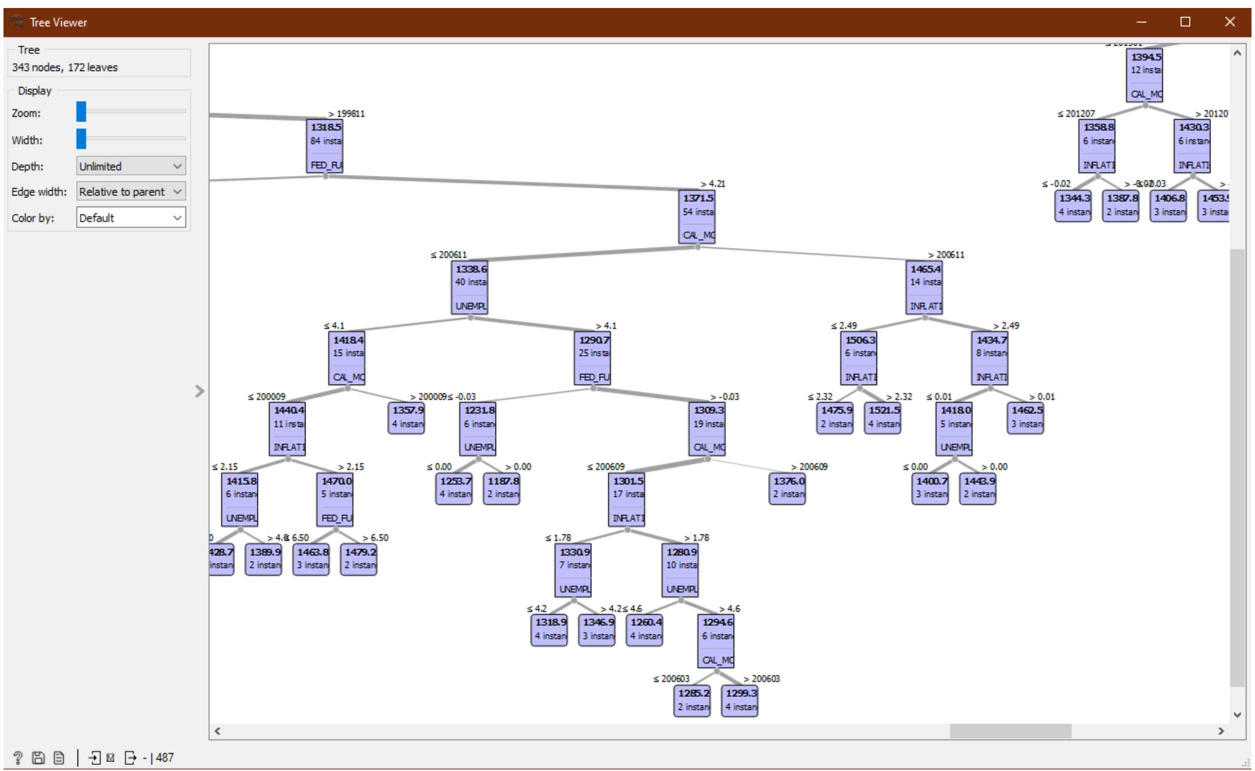

STK_INDEX
Source
File: DM_STK_INDEX.csv
Reload
URL:
Info
487 instance(s)
18 feature(s) (no missing values)
Data has no target variable.
0 meta attribute(s)
Columns (Double click to edit)
Name Type Role Values
1 CAL_MONTH_KEY N numeric feature
2 INDEX_NAME C categorical feature S&P 500
3 AVG_ADJ_CLOSE N numeric feature
4 FED_FUNDS N numeric feature
5 FED_FUNDS_LAG1 N numeric feature
6 FED_FUNDS_CHG N numeric feature
7 FED_FUNDS_PCT_CHG N numeric feature
8 FED_FUNDS_PCT_CHG_SMA N numeric feature
9 INFLATION N numeric feature
10 INFLATION_LAG1 N numeric feature
11 INFLATION_CHG N numeric feature
12 INFLATION_PCT_CHG N numeric feature
13 INFLATION_PCT_CHG_SMA N numeric feature
14 UNEMPLOYMENT N numeric feature
15 UNEMPLOYMENT_LAG1 N numeric feature
16 UNEMPLOYMENT_CHG N numeric feature
17 UNEMPLOYMENT_PCT_CHG N numeric feature
18 UNEMPLOYMENT_PCT_CHG_SMA N numeric feature
Reset
Apply
Browse documentation datasets
487



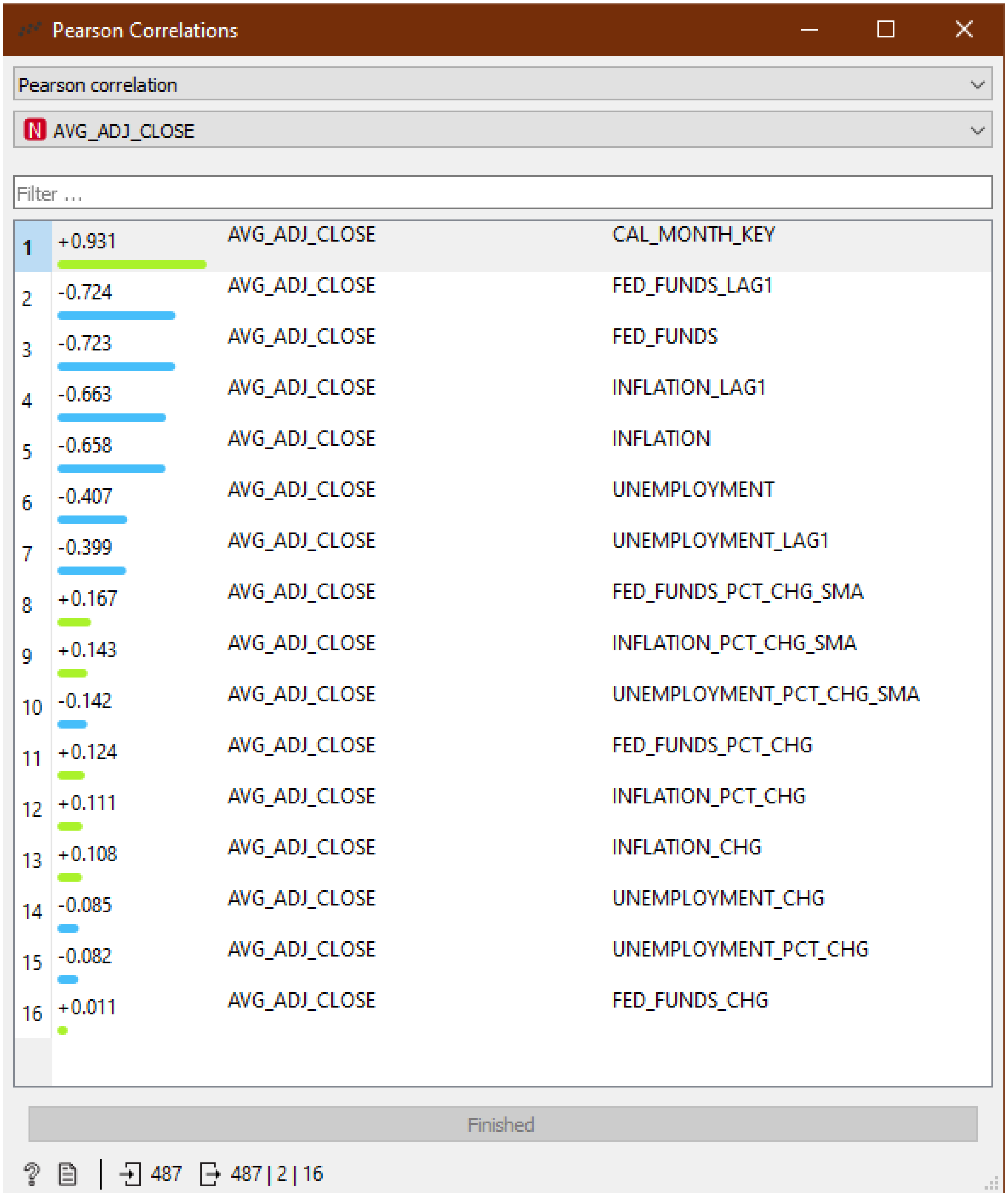
	CAL_MONTH_KEY	INDEX_NAME	AVG_ADJ_CLOSE	FED_FUNDS	FED_FUNDS_LAG1	FED_FUNDS_CHG	FED_FUNDS_PCT_CHG	FED_FUNDS_PCT_CHG_SMA	INFLATION	INFLATION_
1	197801	S&P 500	90.25	6.70	6.70	0.00	0.00	0.00	5.70	
2	197802	S&P 500	88.98	6.78	6.70	0.08	0.01	0.01	5.57	
3	197803	S&P 500	88.82	6.79	6.78	0.01	0.00	0.00	5.55	
4	197804	S&P 500	92.71	6.89	6.79	0.10	0.01	0.01	5.71	
5	197805	S&P 500	97.41	7.36	6.89	0.47	0.07	0.03	5.90	
6	197806	S&P 500	97.66	7.60	7.36	0.24	0.03	0.04	5.91	
7	197807	S&P 500	97.19	7.81	7.60	0.21	0.03	0.04	5.99	
8	197808	S&P 500	103.92	8.04	7.81	0.23	0.03	0.03	6.04	
9	197809	S&P 500	103.86	8.45	8.04	0.41	0.05	0.04	6.22	
10	197810	S&P 500	100.58	8.96	8.45	0.51	0.06	0.05	6.55	
11	197811	S&P 500	94.71	9.76	8.96	0.80	0.09	0.07	6.65	
12	197812	S&P 500	96.11	10.03	9.76	0.27	0.03	0.06	6.71	
13	197901	S&P 500	99.71	10.07	10.03	0.04	0.00	0.04	7.03	
14	197902	S&P 500	98.23	10.06	10.07	-0.01	0.00	0.01	7.22	
15	197903	S&P 500	100.11	10.09	10.06	0.03	0.00	0.00	7.25	
16	197904	S&P 500	102.07	10.01	10.09	-0.08	-0.01	0.00	7.26	
17	197905	S&P 500	99.73	10.24	10.01	0.23	0.02	0.01	7.35	
18	197906	S&P 500	101.73	10.29	10.24	0.05	0.00	0.01	7.40	
19	197907	S&P 500	102.71	10.47	10.29	0.18	0.02	0.02	7.50	
20	197908	S&P 500	107.36	10.94	10.47	0.47	0.04	0.02	7.68	
21	197909	S&P 500	108.60	11.43	10.94	0.49	0.04	0.04	7.80	
22	197910	S&P 500	104.47	13.77	11.43	2.34	0.20	0.10	7.89	
23	197911	S&P 500	103.66	13.18	13.77	-0.59	-0.04	0.07	7.89	


Select Columns
Ignored
Filter
Features
Filter
CAL_MONTH_KEY
INDEX_NAME
FED_FUNDS
FED_FUNDS_LAG1
INFLATION
INFLATION_LAG1
FED_FUNDS_PCT_CHG
UNEMPLOYMENT_PCT_CHG_SMA
UNEMPLOYMENT_PCT_CHG
INFLATION_PCT_CHG
FED_FUNDS_PCT_CHG_SMA
INFLATION_PCT_CHG_SMA
UNEMPLOYMENT
Target
AVG_ADJ_CLOSE
Metas
INFLATION_CHG
UNEMPLOYMENT_CHG
FED_FUNDS_CHG
Reset
Ignore new variables by default
Send Automatically
487 | -
487 | 14





Then I ran some Pearson correlations to see if any useful insights could be gleaned. The FED_FUNDS_LAG1 had the strongest correlation, but it was not strong enough to be considered reliable. Just for kicks, I re-ran the decision tree using FED_FUNDS_LAG1 as the only meta, but the results were essentially the same as the other trees. I also tried ignoring all features except CAL_MONTH_KEY, but the tree results were still essentially the same.



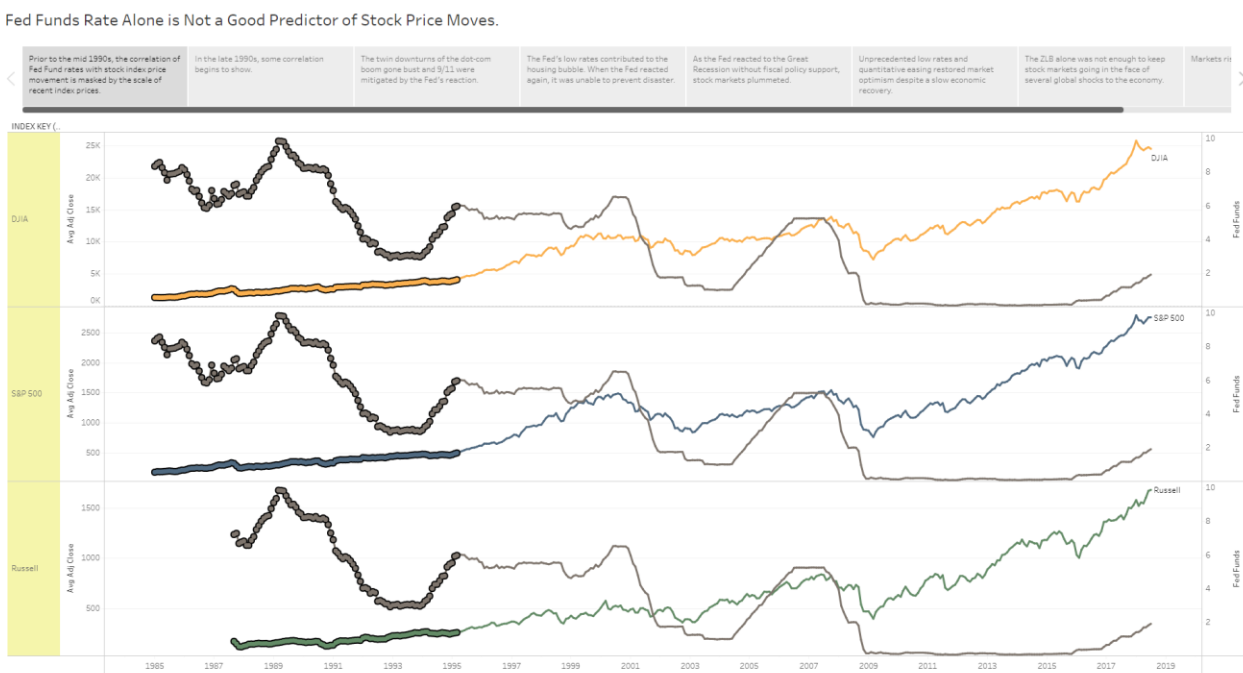
1	+0.931	AVG_ADJ_CLOSE	CAL_MONTH_KEY
2	-0.724	AVG_ADJ_CLOSE	FED_FUNDS_LAG1
3	-0.723	AVG_ADJ_CLOSE	FED_FUNDS
4	-0.663	AVG_ADJ_CLOSE	INFLATION_LAG1
5	-0.658	AVG_ADJ_CLOSE	INFLATION
6	-0.407	AVG_ADJ_CLOSE	UNEMPLOYMENT
7	-0.399	AVG_ADJ_CLOSE	UNEMPLOYMENT_LAG1
8	+0.167	AVG_ADJ_CLOSE	FED_FUNDS_PCT_CHG_SMA
9	+0.143	AVG_ADJ_CLOSE	INFLATION_PCT_CHG_SMA
10	-0.142	AVG_ADJ_CLOSE	UNEMPLOYMENT_PCT_CHG_SMA
11	+0.124	AVG_ADJ_CLOSE	FED_FUNDS_PCT_CHG
12	+0.111	AVG_ADJ_CLOSE	INFLATION_PCT_CHG
13	+0.108	AVG_ADJ_CLOSE	INFLATION_CHG
14	-0.085	AVG_ADJ_CLOSE	UNEMPLOYMENT_CHG
15	-0.082	AVG_ADJ_CLOSE	UNEMPLOYMENT_PCT_CHG
16	+0.011	AVG_ADJ_CLOSE	FED_FUNDS_CHG

I have not yet taken the Data Mining course, so I am not completely certain how to interpret the decision tree results. However, the large number of leaf nodes seems to suggest none of the chosen economic data are reliable predictors of S&P 500 price movement. The low correlations are certainly evidence of a weak predictive model. These observations strengthen the argument for incorporating other indicators, such as a “Fed Sentiment” indicator, into the model. In my experience, the market usually reacts to Fed comments and other indicators of the Fed’s mindset, which always precede Fed actions.

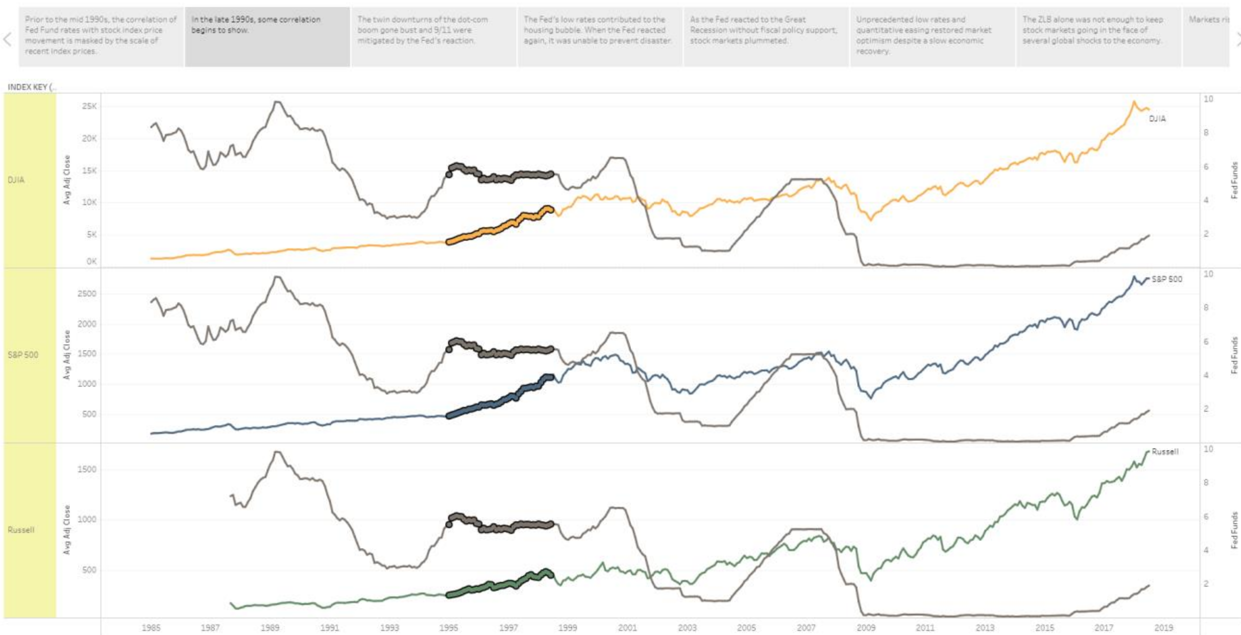
Storytelling

For the storytelling requirement, I created a visualization in Tableau using the story telling feature and then [published it to Tableau Public](). Screenshots are included herein should the published workbook not stand the test of time. I would like to have shown the monthly data in more detail to reveal more volatility in the stock indexes. However, with decades of data, the story would have become a saga, and the theme would have been overwhelmed by detail. Despite the smoothing, some coarse correlations (or lack thereof) are visible that reveal an interesting story.

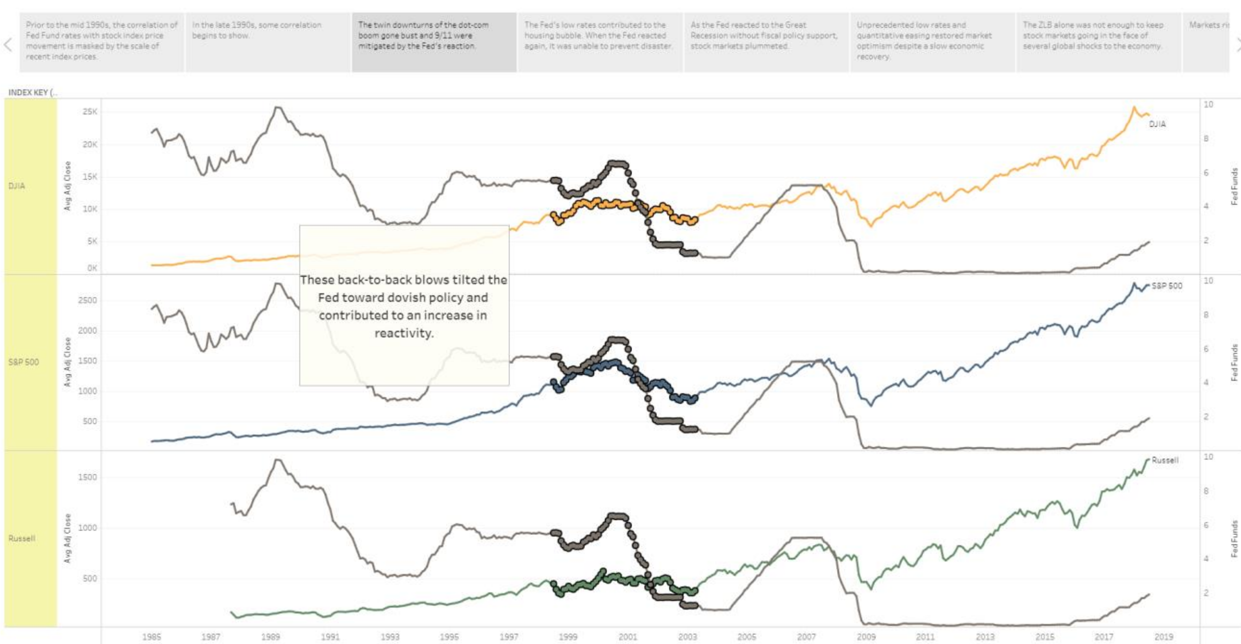
Rather than incorporate all of the candidate predictors, I focused on the Fed Funds rate. Since the data mining effort revealed the strongest correlation to be with the Fed Funds rate, I thought it provided the best opportunity for a useful story to be revealed. In a longer semester, I would have had time to explore the story more fully with multiple predictors.



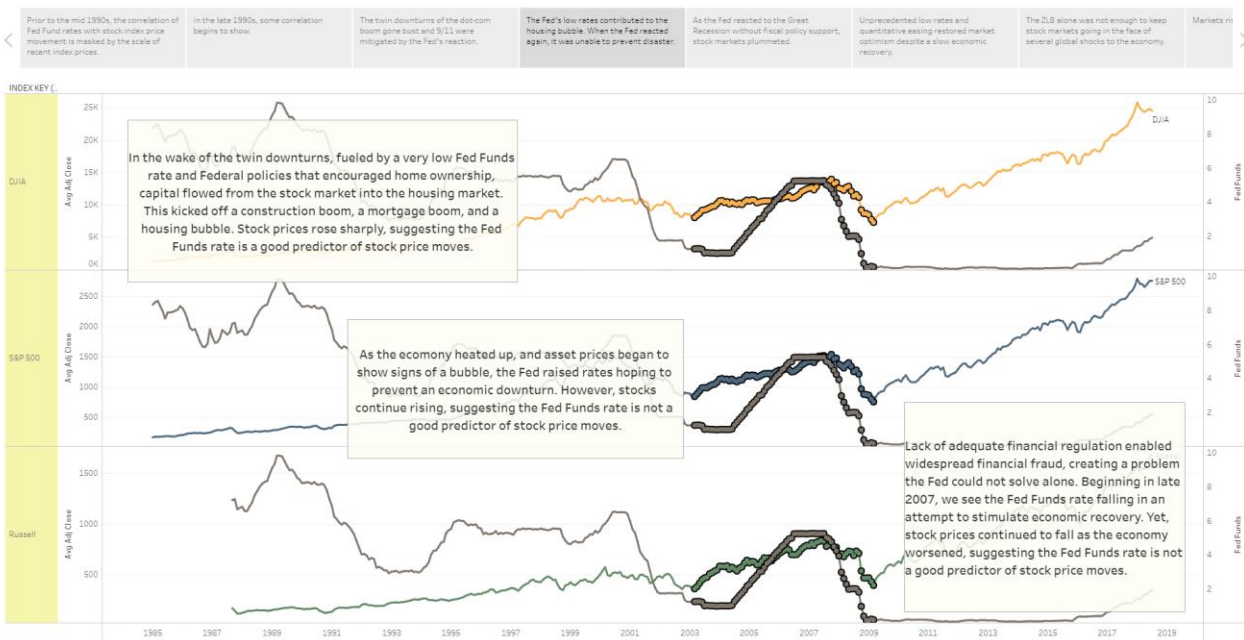
Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



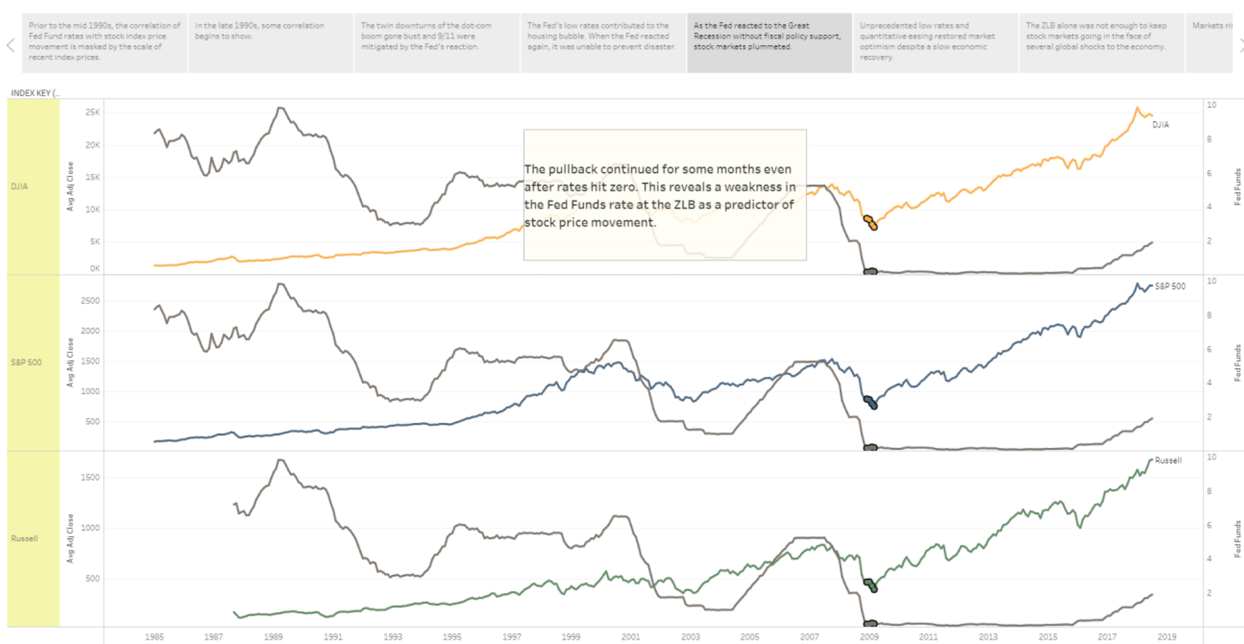
Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



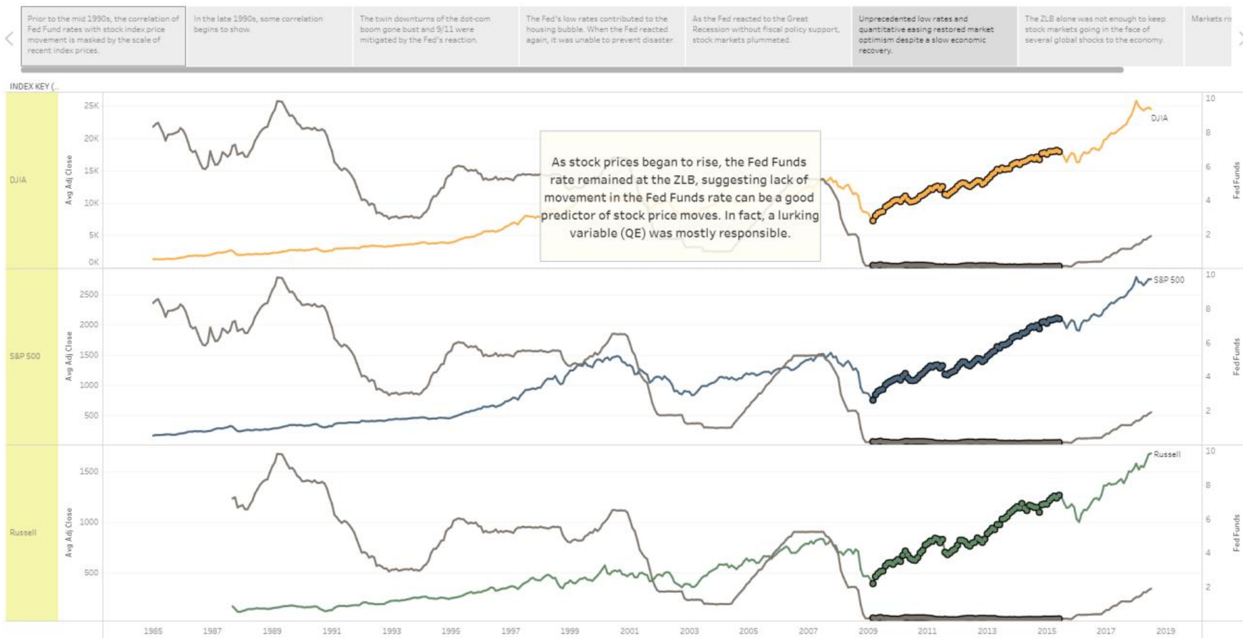
Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



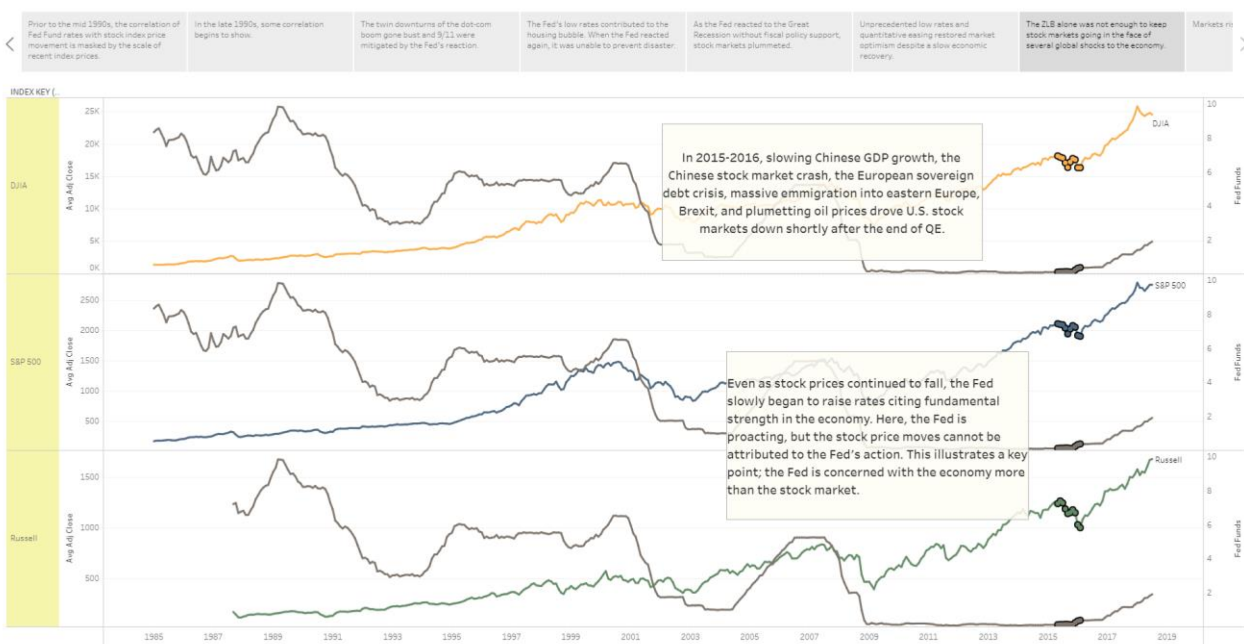
Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



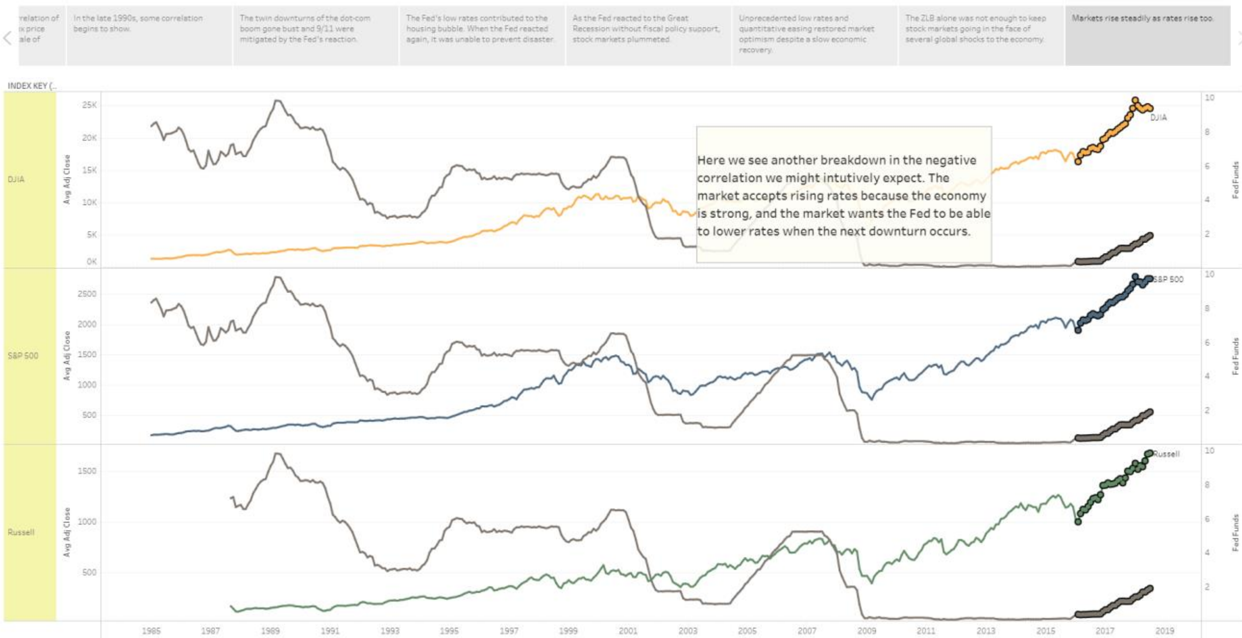
Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



Fed Funds Rate Alone is Not a Good Predictor of Stock Price Moves.



The key take-aways are:

1. The Fed Funds rate alone is not a good predictor of stock index price movement. This visualization corroborates the data mining results.
2. Other Fed tools, like Quantitative Easing (QE), can have even greater influence on stock markets than rate policy.
3. The influence of Fed Funds rate movement can only be properly interpreted by considering other factors such as inflation rates, employment rates, QE activity, world events, financial markets outside the U.S., and so on.

Conclusions

This project was valuable to me because I learned much about data warehousing. But the value was increased by taking the granular facts all the way through mining and visualization. I can now appreciate the importance of schema design and ETL processes for a quality finished product. And I see how difficult it would be to build a reliable prediction model for anything of even modest complexity.

Perhaps the most valuable lesson is that a model is only as good as its data. Without the right inputs, everything else is a waste of time. In my model, a key data set was missing. The Fed's communication precedes and has more influence on stock market psychology than Fed actions (such as adjusting the Fed Funds rate). Without this (and perhaps other) key data, my model is a weak predictor of stock index price movement.