Final Term Project for Database Foundations

Name - Navarurh Kumar
NetID - NXK180010
Date - 12/11/2018

Objective – Answer business questions by utilizing provided data to gain information.

Technology Used

i. Operating System – Ubuntu 16.04 LTS
ii. Database – MySQL
iii. Languages – Python2.7(for data cleaning), R(for data cleaning and data loading to MySQL)
iv. Editors - Markdown (for the final report)

Overall Process Flow

And Discrepancies in the Data and Fixes

- **Data Cleaning**
  - One of the first steps in cleaning the data was establishing the format in which the data was provided
    - The 4 files were `\t` (tab) separated
    - The carriage retrun for eol (end of line) was `\n`
    - The data in `dataset3` had 3 extra tab spaces at the end of each line
      - Removed these as the first step in the cleaning
    - There was no quoting used for any of the fields
  - After this the data was loaded into R using a tab seaparated CSV read
    - This threw new issues with *dataset3*
    - There were a lot of special characters that were causing the file to get truncated before being read fully
    - Implemented a encoding enforcer in Python to remove these special characters
  - Progressing from the data load the next step was to separate the files into the 10 normalized tables as given in the data dictionary
    - The data was sanitized based on the constrainsts provided as part of the data dictionary
    - The `invoice` table had multiple entries for a single `inv_num`
      - Removed the second listing in each of these cases based on an ascending order for the cust_code
    - The `vendor` table had an empty duplicate for `vend_id=15` with no data in the `vend_name` and `vend_street` fields
      - Removed the empty row from the dataset
    - Changed `employee_id` to `emp_num` in `invoice` table to maintain uniformity in naming
    - Changed `emp_num` to `supv_emp_num` in `department` table
    - Fixed datetime field `emp_hire_date` in `employee` table
    - Fixed datetime field `inv_date` in `invoice` table
    - Fixed datetime field `sal_from` in `salary_history` table
    - Fixed datetime field `sal_end` in `salary_history` table
    - `sal_end` had rows with `-` as the data in it was defaulted to `NA` in R and consequently `NULL` in MySQL
    - There was a column constraint on `sal_end` where it was not allowed to hold `NULL` values
      - Removed said table constraint when creating the table as there were quite a few rows with `NULL` data in them
    - Created the following primary keys
      - In table `salary history` : `emp_num,sal_from`
      - In table `supplies` : `prod_sku`
      - In table `line` : `inv_num, prod_sku`
  - We can now move on to answering the query questions and the prediction analysis
- **Exploratory Data Analysis**

  > The EDA reveals characteristic numbers related to the database, helping in understanding the data and quickly validating more complex queries.

  - Table: `brand`
    - There are 9 unique brands
    - The split is on `brand_type` - 4 contractors, 2 Premium and 3 Value
  - Table: `customer`
    - 1362 unique customers present in the database
    - On average a customer has a balance amount of $578
    - 25% of the customers are from the state of New York or Philadelphia
  - Table: `department`
    - There are 8 unique departments in the firm
    - Each department has their own unique supervisor
  - Table: `employee`
    - There are 363 employees at the firm
    - The `TRUCKING` department employs the most people with `PURCHASING` the least with 25

- Table: `invoice`
  - There are 1347 unique invoices on record
  - Surprisingly each invoice is mapped to a unique customer
  - On average an invoice totals $204
- Table: `line`
  - There are 1347 unique items in the `line` table
  - Each line item is associated to a single invoice, again quite aberrant
- Table: `product`
  - There are 252 unique products
  - 146 of those are categorized as `Interior` under `prod_type`
  - 150 of the total are categorized as `Solvent` under `prod_base`
  - And 176 of the total are categorized as `Top Coat` under `prod_category`
- Table: `salary_history`
- Table: `supplies`
  - The `supplies` table it appears creates a mapping between the `vendor` and `product` tables
- Table: `vendor`
  - There are 22 unique vendors in the database
  - There are 3 vendors from each Philadelphia, Michigan and Vermont with most other states having a single vendor

One of the big findings here was that there are certain `prod_sku` 's that are present in the `line` table but do not exist in the `product` table. A list is provided below -

```
select distinct(prod_sku) from line where prod_sku not in (select prod_sku from product);
```

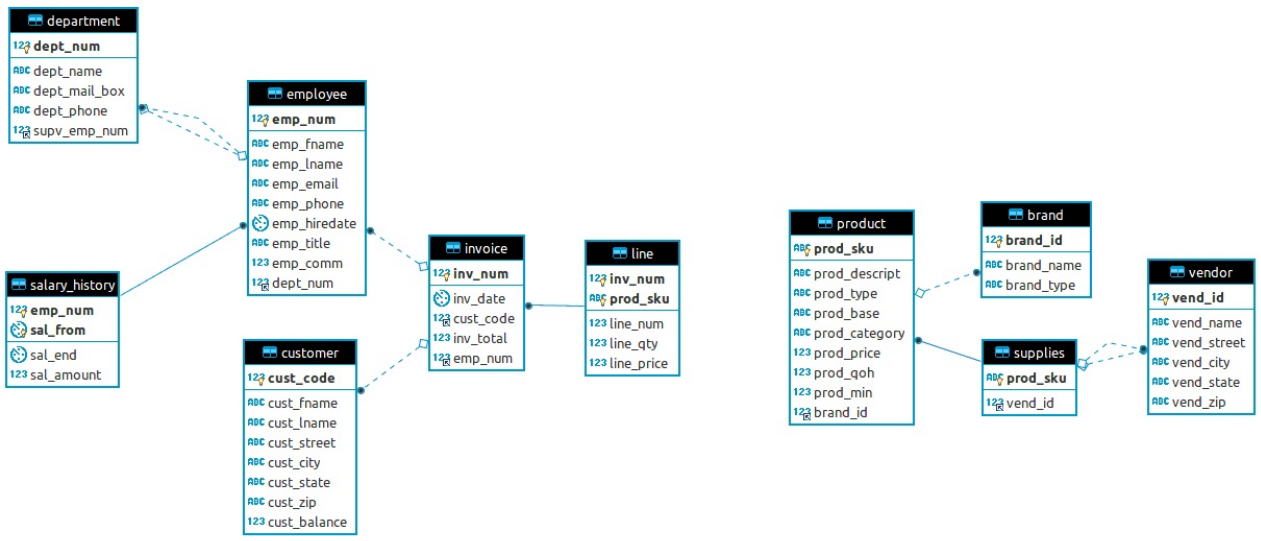
4 rows in set (0.00 sec)
Displaying all rows

prod_sku
2233-GJH
3393-AZQ
5379-BLX
6358-UST

So we see that there are 25 product SKUs that are not present in the `product` table but have an entry made against them in the `line` table.

- **ER Diagram**

Based on our EDA we see that there is no clear connection between the `line` and `product` tables leading to a split ERD with 2 silos.



Query Questions and Solutions

1. Write a query to display the current salary for each employee in department 300. Assume that only current employees are kept in the system, and therefore the most current salary for each employee is the entry in the salary history with a NULL end date. Sort the output in descending order by salary amount.

```
select * from
(
    select
    a.dept_num as department_number,
    a.emp_num as employee_number,
    a.emp_fname as employee_first_name,
    a.emp_lname as employee_last_name,
    b.sal_amount as latest_salary
    from employee as a
    left join salary_history as b
    on a.emp_num = b.emp_num
    where a.dept_num = 300
    and b.sal_end is null
) x
group by x.department_number, x.employee_number, x.employee_first_name, x.employee_last_name, x.latest_salary
order by x.latest_salary desc;
```

25 rows in set (0.00 sec)
Displaying first 10 rows

department_number	employee_number	employee_first_name	employee_last_name	latest_salary
300	83746	SEAN	RANKIN	95550
300	84328	FERN	CARPENTER	94090
300	83716	HENRY	RIVERA	85920
300	84432	MERLE	JAMISON	85360
300	83902	ROCKY	VARGAS	79540
300	83695	CARROLL	MENDEZ	79200
300	84500	CHRISTINE	WESTON	78690
300	84594	ODELL	TIDWELL	77400
300	83910	LAUREN	AVERY	76110
300	83359	MERLE	WATTS	72240

2. Write a query to display the starting salary for each employee. The starting salary would be the entry in the salary history with the oldest salary start date for each employee. Sort the output by employee number.

```
select
    b.emp_num as employee_number,
    b.emp_fname as employee_first_name,
    b.emp_lname as employee_last_name,
    x.sal_amount as first_salary
from
(   select b.*,
    (
        select count(*)
        from salary_history a
        where a.emp_num = b.emp_num
        and a.sal_from <= b.sal_from
        order by a.emp_num,a.sal_from
    ) as ranked
    from salary_history b
    order by b.emp_num,b.sal_from
)x
left join employee as b
on x.emp_num = b.emp_num
where x.ranked = 1
order by b.emp_num;
```

363 rows in set (0.12 sec)
Displaying first 10 rows

employee_number	employee_first_name	employee_last_name	first_salary
83304	TAMARA	MCDONALD	19770
83308	CONNIE	LOVE	11230
83312	ROSALBA	BAKER	39260
83314	CHAROLETTE	DAVID	15150
83318	DARCIE	PECK	22330
83321	ANGELINA	FARMER	18250
83332	WILLARD	LONG	23380
83341	CHRISTINE	CORTEZ	14510
83347	QUINTIN	WINN	17010
83349	JENNIFFER	SINGH	21220

3. Write a query to display the invoice number, line numbers, product SKUs, product descriptions, and brand ID for sales of sealer and top coat products of the same brand on the same invoice.

```
select
distinct
l1.inv_num,
l1.line_num,
p1.prod_sku,
p1.prod_descript,
p1.prod_category,
l2.line_num,
p2.prod_sku,
p2.prod_descript,
p2.prod_category,
p1.brand_id
from (line as l1 left join product as p1 on l1.prod_sku = p1.prod_sku)
inner join (line as l2 left join product as p2 on l2.prod_sku = p2.prod_sku)
on l1.inv_num = l2.inv_num
where p1.prod_category = 'Sealer'
and p2.prod_category = 'Top Coat'
and p1.brand_id = p2.brand_id;
```

Empty set (0.00 sec)

4. The Binder Prime Company wants to recognize the employee who sold the most of their products during a specified period. Write a query to display the employee number, employee first name, employee last name, e-mail address, and total units sold for the employee who sold the most Binder Prime brand products between November 1, 2015, and December 5, 2015. If there is a tie for most units sold, sort the output by employee last name.

```
select
e.emp_num,
e.emp_fname,
e.emp_lname,
e.emp_email,
y.line_tot
from employee as e
left join
(
    select a.emp_num,sum(b.line_qty) as line_tot
    from invoice as a
    left join line as b
    on a.inv_num = b.inv_num
    left join product as c
    on b.prod_sku = c.prod_sku
    join brand as d
    on c.brand_id = d.brand_id
    where d.brand_name = 'BINDER PRIME'
    and a.inv_date >= '2015-11-01'
    and a.inv_date <= '2015-12-05'
    group by a.emp_num
)y
on e.emp_num = y.emp_num
where y.line_tot =
(
    select max(x.line_tot) from
(
    select a.emp_num,sum(b.line_qty) as line_tot
    from invoice as a
    left join line as b
    on a.inv_num = b.inv_num
    left join product as c
    on b.prod_sku = c.prod_sku
    join brand as d
    on c.brand_id = d.brand_id
    where d.brand_name = 'BINDER PRIME'
    and a.inv_date >= '2015-11-01'
    and a.inv_date <= '2015-12-05'
    group by a.emp_num
)x
);
```

Empty set (0.01 sec)

5. Write a query to display the customer code, first name, and last name of all customers who have had at least one invoice completed by employee 83649 and at least one invoice completed by employee 83677. Sort the output by customer last name and then first name.

```
select
a.cust_code,
a.cust_fname,
a.cust_lname,
b.emp_num
from customer as a
left join invoice as b
on a.cust_code = b.cust_code
where a.cust_code in
(select cust_code from invoice where emp_num = 83649)
and b.emp_num = 83677;
```

Empty set (0.00 sec)

6. LargeCo is planning a new promotion in Alabama (AL) and wants to know about the largest purchases made by customers in that state. Write a query to display the customer code, customer first name, last name, full address, invoice date, and invoice total of the largest purchase made by each customer in Alabama. Be certain to include any customers in Alabama who have never made a purchase (their invoice dates should be NULL and the invoice totals should display as 0).

```
select
a.cust_code,
a.cust_fname,
a.cust_lname,
concat(a.cust_street,', ',a.cust_city,', ',a.cust_state,', ',a.cust_zip) as cust_address,
b.inv_date,
b.inv_total
from customer as a
left join invoice as b
on a.cust_code = b.cust_code
where a.cust_state = 'AL'
and b.inv_total = (select max(inv_total) from invoice as c where c.cust_code = a.cust_code)
union
select
d.cust_code,
d.cust_fname,
d.cust_lname,
concat(d.cust_street,', ',d.cust_city,', ',d.cust_state,', ',d.cust_zip) as cust_address,
NULL,
0
from customer as d
left join invoice as e
on d.cust_code = e.cust_code
where d.cust_state = 'AL'
and d.cust_code not in (select cust_code from invoice);
```

50 rows in set (0.01 sec)
Displaying first 10 rows and an 11th row with null invoice total data

cust_code	cust_fname	cust_lname	cust_address	inv_date	inv_total
89	MONICA	CANTRELL	697 ADAK CIRCLE, Loachapoka, AL, 36865	2014-01-14 00:00:00	314
152	LISETTE	WHITTAKER	339 NORTHPARK DRIVE, Montgomery, AL, 36197	2013-11-21 00:00:00	139
169	ROSS	LANG	1991 EASTWIND COURT, Higdon, AL, 35979	2013-11-16 00:00:00	177
188	LUANNE	GOODWIN	293 KIANA AVENUE, Pinegrove, AL, 36507	2013-08-13 00:00:00	202
218	LUPE	SANTANA	1292 WEST 70TH PLACE, Phenix City, AL, 36867	2013-11-02 00:00:00	270
219	CATHI	WHITEHEAD	760 WOODCLIFF DRIVE, Huntsville, AL, 35893	2013-11-21 00:00:00	273
286	JEANNE	STEINER	1974 SCHUSS DRIVE, Carrollton, AL, 35447	2013-03-22 00:00:00	226
295	DORTHY	AUSTIN	829 BIG BEND LOOP, Diamond Shamrock, AL, 36614	2013-11-03 00:00:00	87
304	GERTRUDE	CONNORS	1042 PLEASANT DRIVE, Georgiana, AL, 36033	2013-12-31 00:00:00	376
364	DELLA	MAYO	543 STELIOS CIRCLE, Birmingham, AL, 35214	2013-07-10 00:00:00	95
393	FOSTER	BERNAL	1299 EAST 3RD AVENUE, Birmingham, AL, 35280	NULL	0

7. One of the purchasing managers is interested in the impact of product prices on the sale of products of each brand. Write a query to display the brand name, brand type, average price of products of each brand, and total units sold of products of each brand. Even if a product has been sold more than once, its price should only be included once in the calculation of the average price. However, you must be careful because multiple products of the same brand can have the same price, and each of those products must be included in the calculation of the brand's average price.

```
select
a.brand_name,
a.brand_type,
x.avg_price,
y.total_sold
from brand as a
left join
(
    select brand_id,avg(prod_price) as avg_price
    from product
    group by brand_id
)x
on a.brand_id = x.brand_id
left join
(
    select brand_id,sum(line_qty) as total_sold
    from product as b
    left join line as c
    on b.prod_sku = c.prod_sku
    group by b.brand_id
)y
on a.brand_id = y.brand_id
where y.total_sold is not null;
```

9 rows in set (0.00 sec)
Displaying all rows

brand_name	brand_type	avg_price	total_sold
FORESTERS BEST	VALUE	21.0000	221
STUTTENFURST	CONTRACTOR	16.4815	385
HOME COMFORT	CONTRACTOR	21.8889	466
OLDE TYME QUALITY	CONTRACTOR	18.4074	398
BUSTERS	VALUE	22.6800	479
LONG HAUL	CONTRACTOR	20.2195	665
VALU-MATTE	VALUE	16.8333	312
BINDER PRIME	PREMIUM	16.1481	377
LE MODE	PREMIUM	19.2500	561

8. The purchasing manager is still concerned about the impact of price on sales. Write a query to display the brand name, brand type, product SKU, product description, and price of any products that are not a premium brand, but that cost more than the most expensive premium brand products.

```
select
a.brand_name,
a.brand_type,
b.prod_sku,
b.prod_descript,
b.prod_price
from brand as a
left join product as b
on a.brand_id = b.brand_id
where a.brand_type <> 'PREMIUM'
and b.prod_price >
(
    select max(d.prod_price)
    from brand as c
    left join product as d
    on c.brand_id = d.brand_id
    where c.brand_type = 'PREMIUM'
);
```

1 row in set (0.00 sec)
Dsiplaying all rows

brand_name	brand_type	prod_sku	prod_descript	prod_price
LONG HAUL	CONTRACTOR	1964-OUT	Fire Resistant Top Coat, for Interior Wood	78

9. Using SQL descriptive statistics functions calculate the value of the following items:

9a. What are the products that have a price greater than $50?

```
select * from product where prod_price > 50;
```

3 rows in set (0.00 sec)
Displaying all rows

prod_sku	prod_descript	prod_type	prod_base	prod_category	prod_price	prod_qoh	prod_min	brand_id
1021-MTI	Elastomeric, Exterior, Industrial Grade, Water Based	Exterior	Water	Top Coat	63	22	25	35
1964-OUT	Fire Resistant Top Coat, for Interior Wood	Interior	Solvent	Top Coat	78	120	10	30
3694-XFJ	Epoxy-Modified Latex, Interior, Semi-Gloss (MPI Gloss Level 5)	Interior	Water	Top Coat	55	39	25	27

9b. What is total value of our entire inventory on hand?

```
select sum(prod_qoh*prod_price) as total_inventory_cost from product;
```

This is with the express assumption that `prod_qoh` means “product quantity on hand”.

1 row in set (0.00 sec)
Displaying all rows

total_inventory_cost
359198

9c. How many customers do we presently have and what is the total of all customer balances?

```
select count(*) as no_of_customers from customer;
```

1 row in set (0.00 sec)
Displaying all rows

no_of_customers
1362

```
select sum(cust_balance) as total_customer_balance from customer;
```

1 row in set (0.00 sec)
Displaying all rows

total_customer_balance
787211

9d. What are to top three states that buy the most product in dollars from the company?

```
select
b.cust_state,
sum(a.inv_total) as total_expenditure_state
from invoice as a
left join customer as b
on a.cust_code = b.cust_code
group by b.cust_state
order by sum(a.inv_total) desc
limit 3;
```

3 rows in set (0.04 sec)
Displaying all rows

cust_state	total_expenditure_state
PA	37893
NY	31979
NC	19305

Predictive Analysis for Revenue

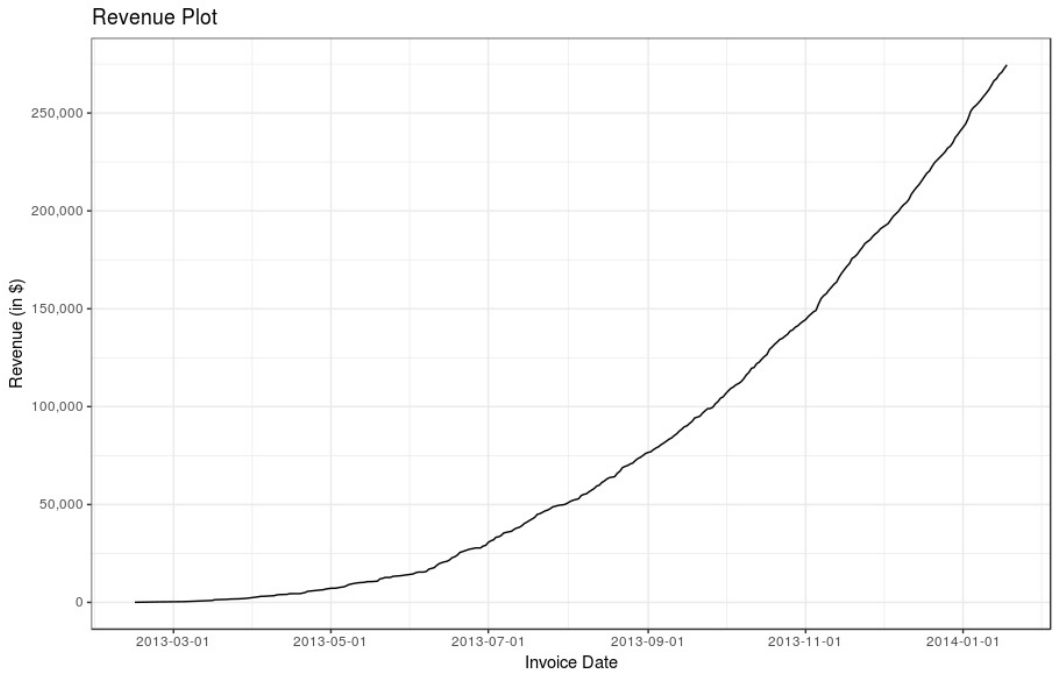
Objective - Predict the revenue for the next year using the data provided

Process
The task is to predict the revenue that the firm will generate in the next year based on the data we have. The process will involve determining the best approach to solve the problem and to provide an easy to understand representation of the prediction.

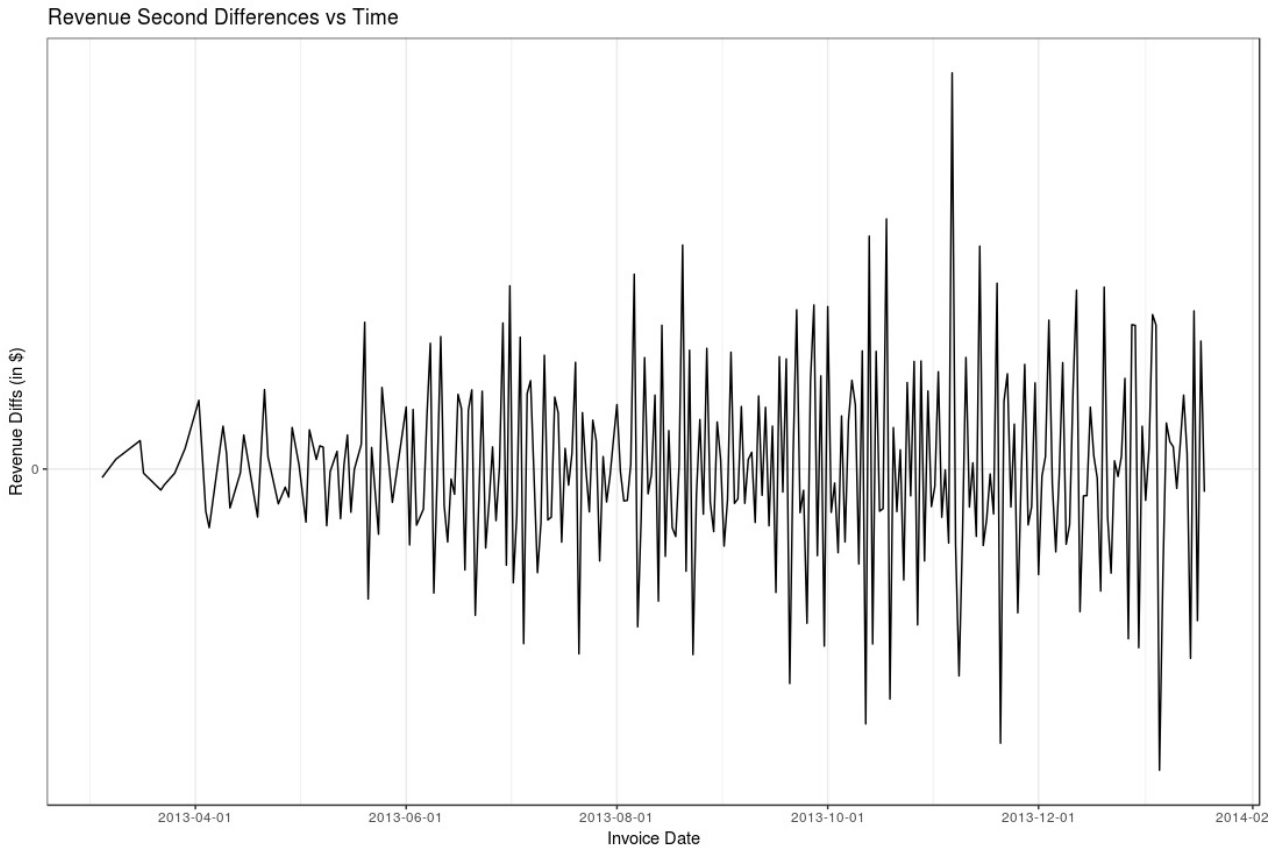
- Looking at the data we can tell that it is a time series problem revolving around revenue that the firm generates
- The first step then is to find the running revenue totals for the given time period - 14th Feb 2013 to 18th Jan 2014
- The total income from invoices is aggregate on a day level to find the earnings for each day
  - `ts_ <- aggregate(data=ts,FUN=sum,inv_total~inv_date)`
- Then we calculate a cumulative sum to find out the exact revenue numbers [assuming that initial revenue was 0 prior to 14th Feb 2013]
  - `ts_$revenue <- cumsum(ts_$inv_total)`
- This finishes our dataset creation with `inv_date` and `revenue` as the required columns
- Next we move on to running the KPSS tests on revenue to see where it is stationary
  - We find that the revenue data is **level stationary** at second differences

```
kpss.test(diff(diff(ts_$revenue)),null="Level")

    KPSS Test for Trend Stationarity

data:  diff(diff(ts_$revenue))
KPSS Trend = 0.077812, Truncation lag parameter = 3, p-value = 0.1

Warning message:
In kpss.test(diff(diff(ts_$revenue)), null = "Trend") :
  p-value greater than printed p-value
```

- The plot for revenue over time is as follows:



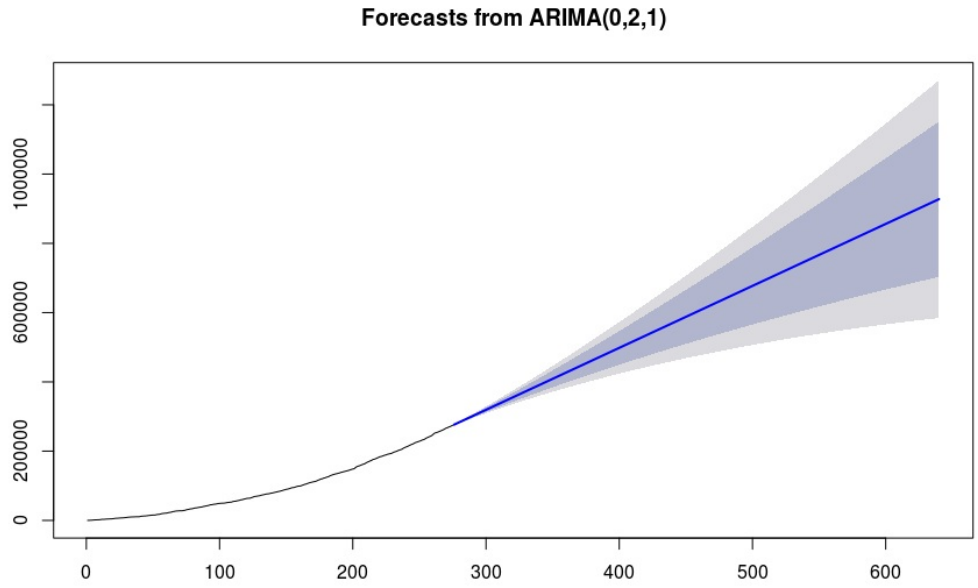
- The plot for second differences of revenue showing stationarity is as follows:



- Next we run an ARIMA model deciding on AR, Diffs and MA values using `auto.arima` in R and plot the prediction for the next 365 days

```
#creating an ARIMA
library(forecast)
auto.arima(ts_$revenue)
model <- arima(ts_$revenue,c(0,2,1))
#setting prediction steps for 365 days
steps <- 365
#predicting the next 365 days of revenue
future <- forecast(model,h=steps)
#plot for predicted revenue
plot(future)
```

- The forecast plot looks like this



- The following table gives us the lower and upper limits for the 80% and 95% confidence intervals [please not that predictions are only being shown for revenue on the first of every month instead of all 365 days of the next year]

Prediction Date	Lower 80% Prediction	Higher 80% Prediction	Lower 95% Prediction	Higher 95% Prediction
2014-02-01	$296053.42	$303286.10	$294139.05	$305200.47
2014-03-01	$338037.38	$361491.53	$331829.44	$367699.47
2014-04-01	$381666.37	$428786.52	$369194.42	$441258.47
2014-05-01	$421564.50	$496234.17	$401800.63	$515998.03
2014-06-01	$460740.06	$567982.57	$432354.68	$596367.95
2014-07-01	$496887.62	$639180.79	$459224.90	$676843.51
2014-08-01	$532585.19	$714407.19	$484459.82	$762532.56
2014-09-01	$566731.59	$791184.76	$507322.43	$850593.93
2014-10-01	$598399.09	$866863.04	$527340.97	$937921.16
2014-11-01	$629783.30	$946402.81	$545979.18	$1030206.93
2014-12-01	$658927.44	$1024604.44	$562138.59	$1121393.30
2015-01-01	$687835.34	$1106620.52	$576989.59	$1217466.26

Insights

- The process shows that the increase in revenue over the last year will follow into the next year
- There will be a steady 8.66% growth in revenue every month (on average) over the next 12 months
- We can expect the revenue at the end of the whole year to atleast double in the next 12 months

Code Used in the Process Flow

The data in BUAN6320-DataSet3.txt had a lot of special characters that were causing issues when importing the data into R for cleaning. The removal of these characters was carried out using a python script to clean out characters by forcing UTF-8 encoding for the file. The script used was

```
x = open("datasets/BUAN6320-DataSet3.txt", "r")
opt = x.read()
x.close()
opt = opt.decode('utf-8','ignore').encode("utf-8")
print opt
o = open("datasets/BUAN6320-DataSet3_pytmp.txt", "w")
o.write(opt)
o.close
```

The data was then split into 10 different tables using the following Rscript

```
setwd("~/documents/assignments/database/project/")

#importing libraries
library(plyr)
library(RMySQL)

#reading in datasets
d1 <- read.csv("datasets/BUAN6320-DataSet1.txt",header = T,stringsAsFactors = F,sep = "\t")
d2 <- read.csv("datasets/BUAN6320-DataSet2.txt",header = T,stringsAsFactors = F,sep = "\t")
d3 <- read.csv("datasets/BUAN6320-DataSet3_pytmp.txt",header = T,stringsAsFactors = F,sep = "\t",fileEncoding="utf-8")
d4 <- read.csv("datasets/BUAN6320-DataSet4.txt",header = T,stringsAsFactors = F,sep = "\t")

#displaying column names for easier subsetting
colnames(d1)
colnames(d2)
colnames(d3)
colnames(d4)

#subsetting columns to form individual tables and manipulating rows to clean data
brand <- d3[c(12,2,9)]
brand <- unique(brand)
customer <- d1[c(2,3,5,6,8,10,12,14)]
customer <- unique(customer)
department <- d4[c(12,13,14,15,16)]
department <- unique(department)
employee <- d4[c(1,2,3,4,5,6,7,8,12)]
employee <- unique(employee)
invoice <- d1[c(13,16,2,9,15)]
invoice <- unique(invoice)
line1 <- d1[c(13,11,7,4,1)]
line2 <- d2[c(12,13,14,15,16)]
line <- rbind(line1,line2)
line <- unique(line)
product <- d3[c(1,4,5,17,7,11,14,16,12)]
product <- unique(product)
salary_history <- d4[c(1,9,10,11)]
salary_history <- unique(salary_history)
supplies <- d3[c(1,13)]
supplies <- unique(supplies)
vendor <- d3[c(13,6,3,10,8,15)]
vendor <- unique(vendor)

#removing duplicated invoice numbers
invoice <- invoice[!duplicated(invoice$INV_NUM),]
#removing empty vendor names
vendor <- vendor[which(vendor$VEND_NAME != ""),]
#changing employee_id to emp_num in invoices table
colnames(invoice)[5] <- "EMP_NUM"
#reformatting dates into the proper YYYY-MM-DD format
employee$EMP_HIREDATE <- as.Date(employee$EMP_HIREDATE, origin = "1900-01-01")
invoice$INV_DATE <- as.Date(invoice$INV_DATE, origin = "1900-01-01")
salary_history$SAL_FROM <- as.Date(salary_history$SAL_FROM, origin = "1900-01-01")
#setting empty sal_end dates to null in salary_history table
salary_history$SAL_END[which(salary_history$SAL_END == " - ")] <- NA
salary_history$SAL_END <- as.integer(salary_history$SAL_END)
salary_history$SAL_END <- as.Date(salary_history$SAL_END, origin = "1900-01-01")

#pushing data to mysql
con <- dbConnect(MySQL(), user='root', password='n', dbname='project', host='localhost')

dbWriteTable(con, name='brand', brand, append=T, row.names = F)
dbWriteTable(con, name='customer', customer, append=T, row.names = F)
dbWriteTable(con, name='department', department, append=T, row.names = F)
dbWriteTable(con, name='employee', employee, append=T, row.names = F)
```

```
dbWriteTable(con, name='invoice', invoice, append=T, row.names = F)
dbWriteTable(con, name='line', line, append=T, row.names = F)
dbWriteTable(con, name='product', product, append=T, row.names = F)
dbWriteTable(con, name='salary_history', salary_history, append=T, row.names = F)
dbWriteTable(con, name='supplies', supplies, append=T, row.names = F)
dbWriteTable(con, name='vendor', vendor, append=T, row.names = F)
```

The table definitions for the normal form as described in the data dictionary was created in MySQL using the following script

```
CREATE DATABASE project;
USE project;

CREATE TABLE IF NOT EXISTS brand
(
    brand_id    bigint(4)   not null,
    brand_name  text(100),
    brand_type  text(20),
    PRIMARY KEY (brand_id)
);

CREATE TABLE IF NOT EXISTS  customer
(
    cust_code       bigint(4)   not null,
    cust_fname      text(20),
    cust_lname      text(20),
    cust_street     text(70),
    cust_city       text(50),
    cust_state      text(2),
    cust_zip        text(5),
    cust_balance    decimal(16) not null,
    PRIMARY KEY (cust_code)
);

CREATE TABLE IF NOT EXISTS  department
(
    dept_num        bigint(4)   not null,
    dept_name       text(50),
    dept_mail_box   text(3),
    dept_phone      text(9),
    supv_emp_num    bigint(4)   not null,
    PRIMARY KEY (dept_num)
);

CREATE TABLE IF NOT EXISTS  employee
(
    emp_num         bigint(4)   not null,
    emp_fname       text(20),
    emp_lname       text(20),
    emp_email       text(25),
    emp_phone       text(20),
    emp_hiredate    datetime not null,
    emp_title       text(45),
    emp_comm        decimal(16) not null,
    dept_num        bigint(4)   not null,
    PRIMARY KEY (emp_num),
    INDEX   idx_dn(dept_num asc),
    INDEX   idx_en(emp_num asc)
);

CREATE TABLE IF NOT EXISTS  invoice
(
    inv_num     bigint(4)   not null,
    inv_date    datetime not null,
    cust_code   bigint(4)   not null,
    inv_total   decimal(16) not null,
    emp_num     bigint(4)   not null,
    PRIMARY KEY (inv_num),
    INDEX   idx_cc(cust_code asc),
    INDEX   idx_in(inv_num asc),
    INDEX   idx_en(emp_num asc)
);

CREATE TABLE IF NOT EXISTS  line
(
    inv_num     bigint(4)   not null,
    line_num    bigint(4)   not null,
    prod_sku    text(15),
    line_qty    bigint(8)   not null,
    line_price  decimal(16) not null
);
```

```
CREATE TABLE IF NOT EXISTS  product
(
    prod_sku        text(15),
    prod_descript   text(255),
    prod_type       text(255),
    prod_base       text(255),
    prod_category   text(255),
    prod_price      decimal(16) not null,
    prod_qoh        decimal(16) not null,
    prod_min        decimal(16) not null,
    brand_id        bigint(4)   not null
);

CREATE TABLE IF NOT EXISTS  salary_history
(
    emp_num     bigint(4)   not null,
    sal_from    datetime not null,
    sal_end     datetime,
    sal_amount  decimal(16) not null
);

CREATE TABLE IF NOT EXISTS  supplies
(
    prod_sku    text(15),
    vend_id     bigint(4)   not null
);

CREATE TABLE IF NOT EXISTS  vendor
(
    vend_id     bigint(4)   not null,
    vend_name   text(255),
    vend_street text(50),
    vend_city   text(50),
    vend_state  text(2),
    vend_zip    text(5),
    PRIMARY KEY (vend_id)
);




ALTER TABLE product
MODIFY COLUMN prod_sku varchar(255);
ALTER TABLE product
ADD PRIMARY KEY (prod_sku);
ALTER TABLE product
ADD FOREIGN KEY (brand_id) REFERENCES brand(brand_id);


ALTER TABLE department
ADD FOREIGN KEY (supv_emp_num) REFERENCES employee(emp_num);


ALTER TABLE employee
ADD FOREIGN KEY (dept_num) REFERENCES department(dept_num);


ALTER TABLE invoice
ADD FOREIGN KEY (cust_code) REFERENCES customer(cust_code),
ADD FOREIGN KEY (emp_num) REFERENCES employee(emp_num);


ALTER TABLE line
MODIFY COLUMN prod_sku varchar(255);
ALTER TABLE line
ADD PRIMARY KEY (prod_sku,inv_num);
ALTER TABLE line
ADD FOREIGN KEY (inv_num) REFERENCES invoice(inv_num);


ALTER TABLE salary_history
ADD PRIMARY KEY (emp_num,sal_from);
ALTER TABLE salary_history
ADD FOREIGN KEY (emp_num) REFERENCES employee(emp_num);


ALTER TABLE supplies
MODIFY COLUMN prod_sku varchar(255);
ALTER TABLE supplies
ADD PRIMARY KEY (prod_sku);
ALTER TABLE supplies
ADD FOREIGN KEY (vend_id) REFERENCES vendor(vend_id);
```

```
ALTER TABLE supplies
ADD FOREIGN KEY (prod_sku) REFERENCES product(prod_sku);
```

The script for the predictive analysis is as follows

```
#This code follows from the previous R-code for generating the datasets, the dataframe names and columns being used are the same and this
 code can be executed in continuation from where the previous code left off

#time series analysis
library(ggplot2)
library(tseries)

#subsetting invoice table into an analytical dataset
ts <- invoice[c(2,4)]
ts <- ts[order(ts$INV_DATE),]
colnames(ts) <- c("inv_date","inv_total")
#aggregating the invoice totals on a day level
ts_ <- aggregate(data=ts,FUN=sum,inv_total~inv_date)
#talking a cumulative sum of invoice totals on a day level to generate the running revenue field
ts_$revenue <- cumsum(ts_$inv_total)

#testing revenue data to check where it is stationary
kpss.test(ts_$revenue,null="Level")
kpss.test(ts_$revenue,null="Trend")
kpss.test(diff(ts_$revenue),null="Level")
kpss.test(diff(ts_$revenue),null="Trend")
kpss.test(diff(diff(ts_$revenue)),null="Level")
kpss.test(diff(diff(ts_$revenue)),null="Trend")

#making a time series plot to check revenue data being stationary
plt_rev <- ggplot(ts_, aes(x=inv_date, y=revenue, group=1)) +
  geom_line() +
  xlab('Invoice Date') +
  ylab('Revenue (in $)') +
  theme_bw() +
  ggtitle('Revenue Plot') +
  scale_x_date(date_breaks = "2 months") +
  scale_y_continuous(labels = scales::comma, breaks=c(0,50000,100000,150000,200000,250000,300000))
plt_rev

revdiffs <- diff(diff(ts_$revenue))
dates <- ts_$inv_date[3:275]

df1 <- as.data.frame(cbind(dates,revdiffs))
df1$dates <- as.Date(df1$dates,origin = '1970-01-01')

plt_rev_diff <- ggplot(data = df1,aes(x=dates, y=revdiffs, group=1)) +
  geom_line() +
  xlab('Invoice Date') +
  ylab('Revenue Diffs (in $)') +
  theme_bw() +
  ggtitle('Revenue Second Differences vs Time') +
  scale_x_date(date_breaks = "2 months") +
  scale_y_continuous(labels = scales::comma, breaks=c(0,50000,100000,150000,200000,250000,300000))
plt_rev_diff

#creating an ARIMA
library(forecast)
auto.arima(ts_$revenue)
model <- arima(ts_$revenue,c(0,2,1))
#setting prediction steps for 365 days
steps <- 365
#predicting the next 365 days of revenue
future <- forecast(model,h=steps)
#plot for predicted revenue
plot(future)
fcast <- as.data.frame(cbind(future$lower,future$upper))
fcast$date <- seq.Date(from = max(ts_$inv_date)+1,to = max(ts_$inv_date)+365,by = "day")
fcast <- fcast[c(5,1,3,2,4)]
fcast$`future$lower.80%` <- round(fcast$`future$lower.80%`,2)
fcast$`future$upper.80%` <- round(fcast$`future$upper.80%`,2)
fcast$`future$lower.95%` <- round(fcast$`future$lower.95%`,2)
fcast$`future$upper.95%` <- round(fcast$`future$upper.95%`,2)
colnames(fcast) <- c("Prediction Date","Lower 80% Prediction","Higher 80% Prediction","Lower 95% Prediction", "Higher 95% Prediction")
dbWriteTable(con, name='forecasts', fcast, append=T, row.names = F)
fcst <- fcast
fcst$`Prediction Date` <- as.character(fcst$`Prediction Date`)
fcst <- fcst[which(substr(fcst$`Prediction Date`,9,10) == '01'),]
fcst$avrev <- (fcst$`Lower 95% Prediction`+fcst$`Higher 95% Prediction`)/2
fcst$diff <- 0
fcst$diff[2:12] <- diff(fcst$avrev)
```

```
fcst$pinc <- fcst$diff/fcst$avrev*100
mean(fcst$pinc)
```