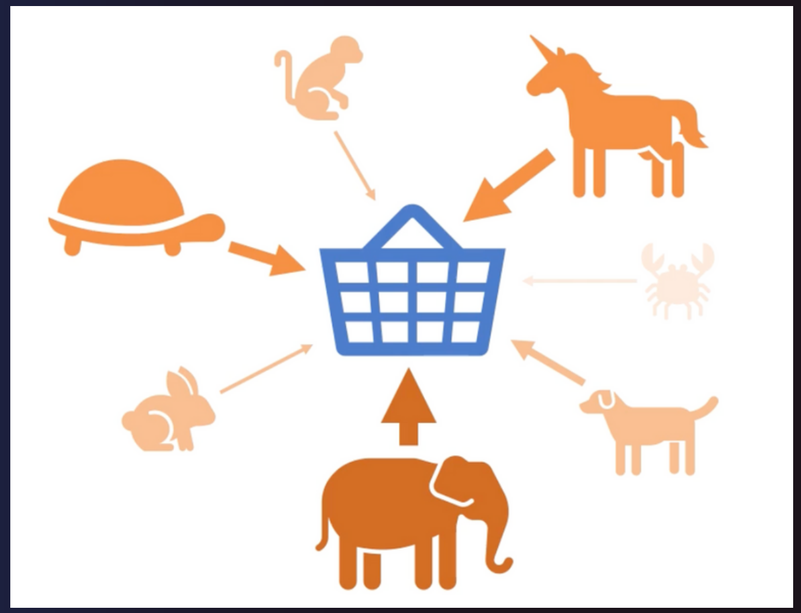
Product Sales Analysis

Analyzing product sales can help you understand how each product contributes to your business and how new product launches impact your overall portfolio.

Some everyday use cases here include:

- Analyzing your sales and revenue by either product or line of business
- Monitor the impact of adding a new product to the portfolio. Is this additive? Does it cannibalize other products?
- Keep an eye on product-specific sales trends so you can understand your business's overall health.



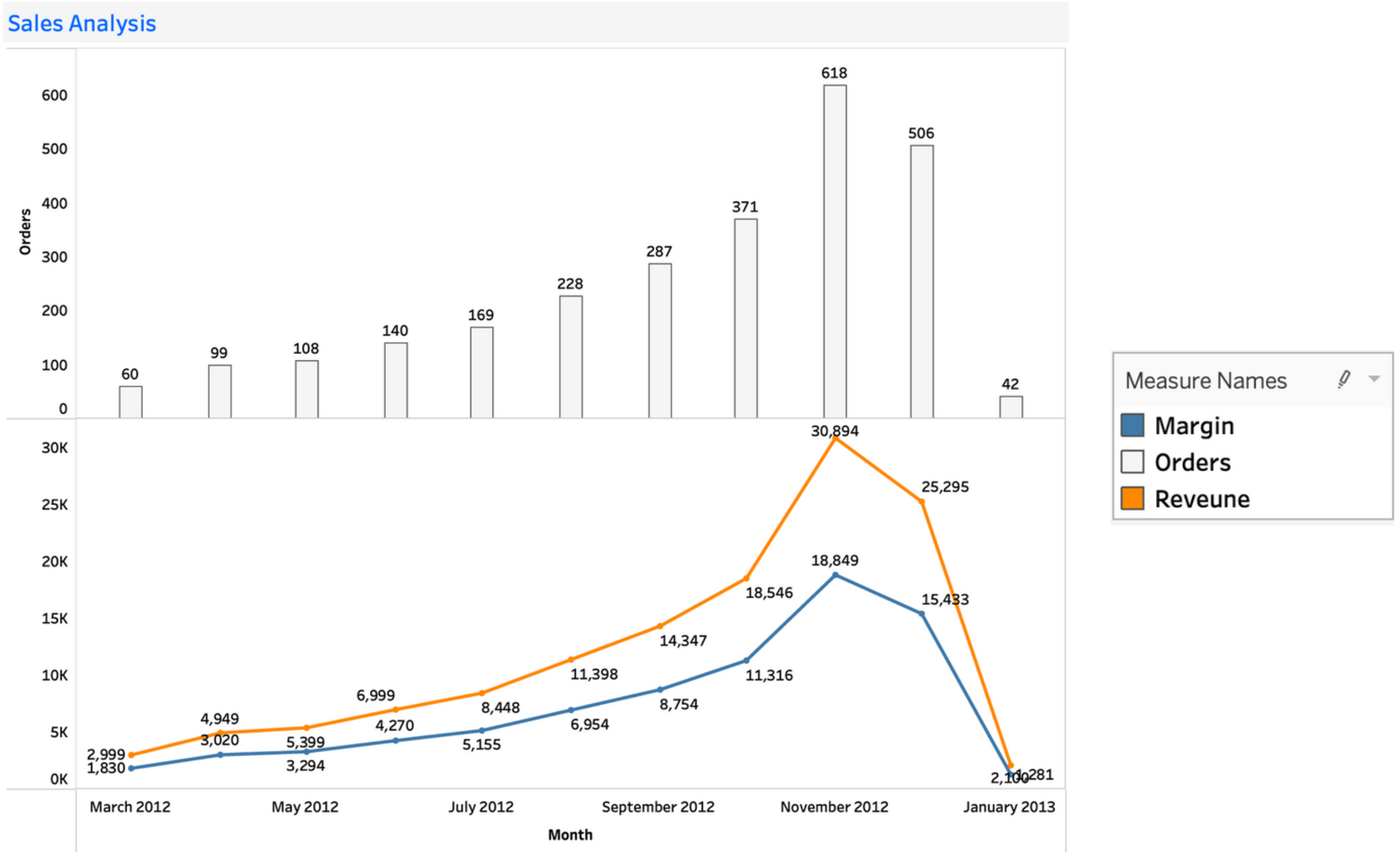
Sales Analysis | SQL

```sql
-- Monthly trends of the orders,
-- revenue generated and the margin
SELECT
    YEAR(created_at) AS Year,
    MONTH(created_at) AS Month,
    COUNT(DISTINCT order_id) AS orders,
    SUM(price_usd) AS reveune,
    SUM(price_usd - cogs_usd) AS margin
FROM orders
WHERE created_at <= '2013-01-04'
GROUP BY 1,2;
```

00% 6:13

Result Grid | Filter Rows: Search | Export

Year	Month	orders	reveune	margin
2012	3	60	2999.40	1830.00
2012	4	99	4949.01	3019.50
2012	5	108	5398.92	3294.00
2012	6	140	6998.60	4270.00
2012	7	169	8448.31	5154.50
2012	8	228	11397.72	6954.00
2012	9	287	14347.13	8753.50
2012	10	371	18546.29	11315.50
2012	11	618	30893.82	18849.00
2012	12	506	25294.94	15433.00
2013	1	42	2099.58	1281.00



Findings:

- Sales are going up from 60, 99 to all the way up to 618 in November of 2012 and we see similar trends with increasing revenue and increasing total margin.
- The data displayed is till Jan 4th, 2013. ***

Impact of New Product Launch | SQL

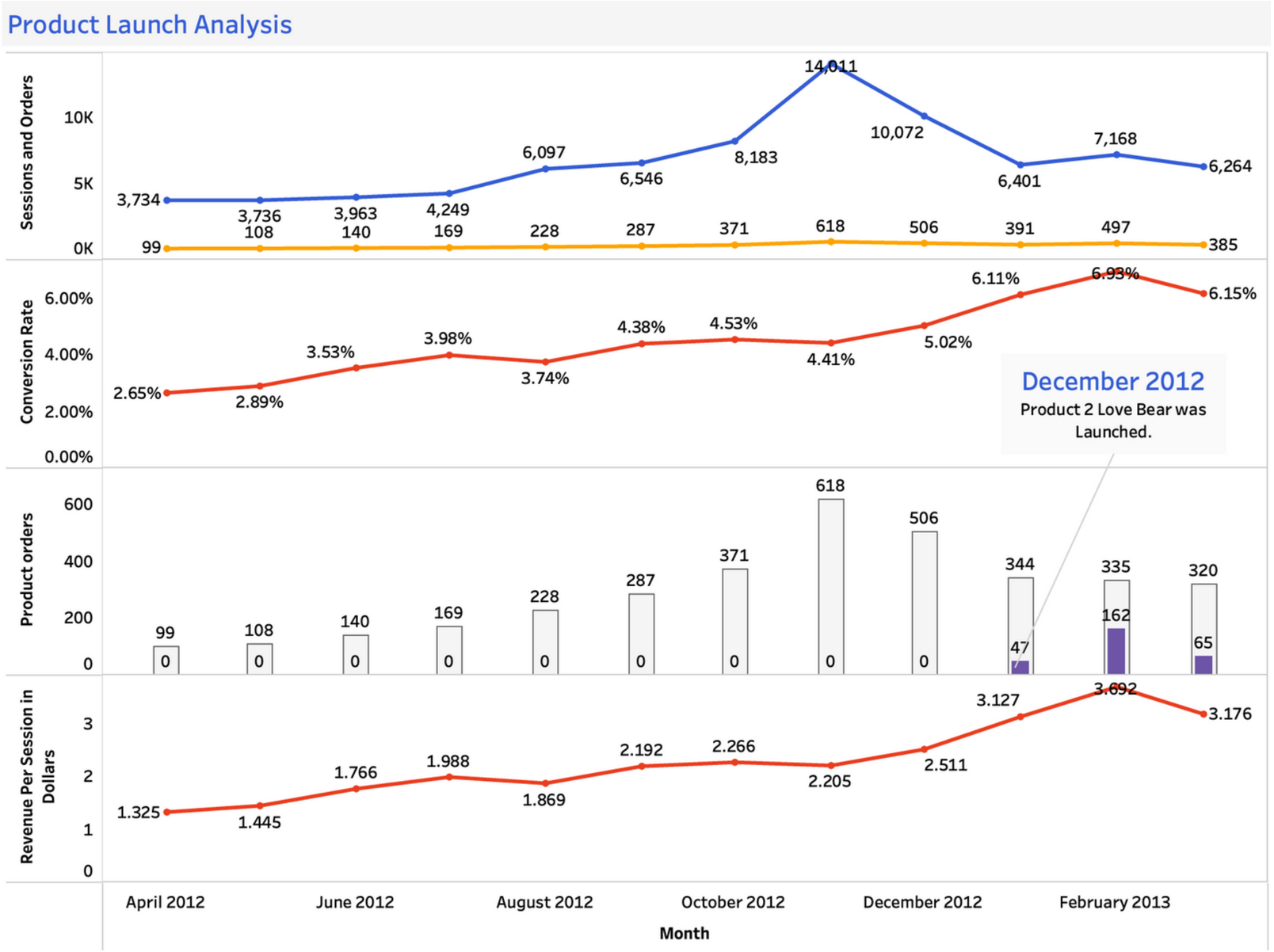
```sql
-- Impact of New Product Launch

SELECT
    YEAR(W.created_at) AS YEAR,
    MONTH(W.created_at) AS MONTH,
    COUNT(DISTINCT W.website_session_id) AS sessions,
    COUNT(DISTINCT O.order_id) AS orders,
    COUNT(DISTINCT O.order_id)/COUNT(DISTINCT W.website_session_id) AS conversion_rate,
    SUM(O.price_usd)/COUNT(DISTINCT W.website_session_id) AS revenue_per_session,
    COUNT(DISTINCT CASE WHEN O.primary_product_id = 1 THEN order_id ELSE NULL END) AS product_one_orders,
    COUNT(DISTINCT CASE WHEN O.primary_product_id = 2 THEN order_id ELSE NULL END) AS product_two_orders
FROM website_sessions AS W
    LEFT JOIN orders AS O ON W.website_session_id = O.website_session_id
WHERE W.created_at > '2012-04-01'
AND W.created_at < '2013-04-04'
GROUP BY 1,2
```

13:32

Result Grid | Filter Rows: Search | Export:

YEAR	MONTH	sessions	orders	conversion_rate	revenue_per_sessi...	product_one_ord...	product_two_ord...
2012	4	3734	99	0.0265	1.325391	99	0
2012	5	3736	108	0.0289	1.445107	108	0
2012	6	3963	140	0.0353	1.765985	140	0
2012	7	4249	169	0.0398	1.988305	169	0
2012	8	6097	228	0.0374	1.869398	228	0
2012	9	6546	287	0.0438	2.191740	287	0
2012	10	8183	371	0.0453	2.266441	371	0
2012	11	14011	618	0.0441	2.204969	618	0
2012	12	10072	506	0.0502	2.511412	506	0
2013	1	6401	391	0.0611	3.127025	344	47
2013	2	7168	497	0.0693	3.692108	335	162
2013	3	6264	385	0.0615	3.176269	320	65
2013	4	899	74	0.0823	4.237219	63	11

Impact of New Product Launch | SQL



Findings:

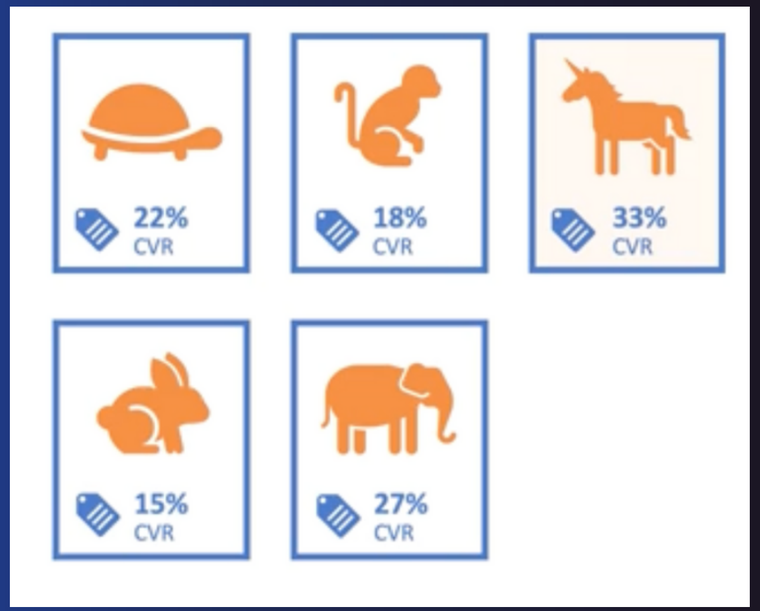
- We see product two getting a lot of sales in February 2013 but reduced here in March.
- Our revenue per session seems like growing pretty strong, and our conversion rates have similarly improved.
- Since the new product launch, the revenue per session exceeded 3 dollars.

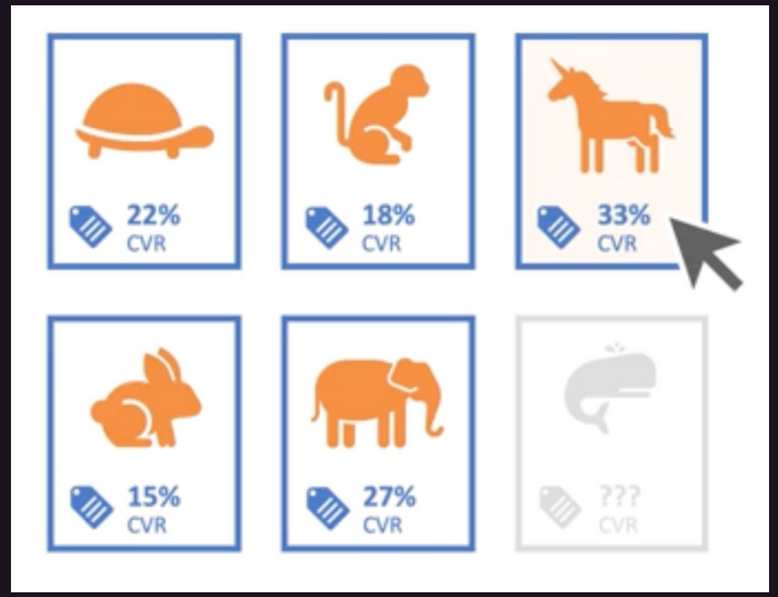
Product Level Website Analysis

Product focus website analysis is all about learning how customers interact with each of your products and then how well each of those products convert your customers.

Some everyday use cases here include:

- Understanding which products generate the most interest on the multi-product showcase pages.
- Analyzing the impact on website conversion when we add a new product to the portfolio.
- Building product-specific conversion funnels to understand whether certain products are converting better than others and where certain products may have trouble in the conversion.





Product Level User Pathing on website | SQL

```sql
-- Product Pathing Analysis
-- Lets look at how many people hit the products page and where they went after wards
-- Need to compare 3 months before and 3 months after product launch
-- Product 2 was launched on Jan 06, 2013

-- Step 1 Identify the product sessions
CREATE TEMPORARY TABLE product_sessions
SELECT
    CASE WHEN W.created_at < '2013-01-06' THEN 'pre_product_launch'
         WHEN W.created_at >= '2013-01-06' THEN 'post_product_launch'
         ELSE 'check_logic' END AS time_line,
    W.website_session_id,
    W.website_pageview_id,
    W.created_at
FROM website_pageviews AS W
WHERE W.created_at < '2013-04-06'
AND W.created_at > '2012-10-06'
AND W.pageview_url = '/products';
SELECT * FROM product_sessions;
```

32:56

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_line	website_session...	website_pageview...	created_at
pre_product_launch	31517	67216	2012-10-06 00:01:26
pre_product_launch	31518	67220	2012-10-06 00:20:27
pre_product_launch	31519	67222	2012-10-06 00:24:31
pre_product_launch	31521	67227	2012-10-06 00:48:54
pre_product_launch	31524	67232	2012-10-06 01:50:14
pre_product_launch	31525	67236	2012-10-06 02:11:18
pre_product_launch	31528	67240	2012-10-06 03:27:07

```sql
-- Step 2 Find the next pageview that occours after the products pageview
CREATE TEMPORARY TABLE next_pv_table
SELECT
    P.time_line,
    P.website_session_id,
    MIN(W.website_pageview_id) AS next_pv_id
FROM product_sessions AS P
    LEFT JOIN website_pageviews AS W
    ON P.website_session_id = W.website_session_id
    AND W.website_pageview_id > P.website_pageview_id -- Very IMP
GROUP BY 1,2;
SELECT * FROM next_pv_table;
-- Next_pv_id Null then website_session_bounced at products_page
```

29:69

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_line	website_session...	next_pv_id
pre_product_launch	31549	67281
pre_product_launch	31551	NULL
pre_product_launch	31552	NULL
pre_product_launch	31559	67293
pre_product_launch	31560	67296
pre_product_launch	31562	67302
pre_product_launch	31563	67305
pre_product_launch	31569	67315
pre_product_launch	31568	67314
pre_product_launch	31570	67319

```sql
-- Step 3 Find the url associated with next_pv_id
CREATE TEMPORARY TABLE next_pageviews
SELECT
    N.time_line,
    N.website_session_id,
    W.pageview_url AS next_pageview_url
FROM next_pv_table AS N
    LEFT JOIN website_pageviews AS W
    ON N.next_pv_id = W.website_pageview_id;
SELECT * FROM next_pageviews;
```

30:81

Result Grid | Filter Rows: Search | Export:

time_line	website_session...	next_pageview_url
post_product_launch	66210	/the-original-mr-fuzzy
post_product_launch	66211	/the-original-mr-fuzzy
post_product_launch	66215	/the-forever-love-bear
post_product_launch	66216	/the-original-mr-fuzzy
post_product_launch	66217	NULL

```sql
-- Step 4 Summarizing the data
SELECT
    time_line,
    COUNT(DISTINCT website_session_id) AS product_sessions,
    COUNT(DISTINCT CASE WHEN next_pageview_url IS NOT NULL THEN website_session_id ELSE NULL END) AS w_next_pg,
    COUNT(DISTINCT CASE WHEN next_pageview_url IS NOT NULL THEN website_session_id ELSE NULL END)/COUNT(DISTINCT website_session_id)  AS pct_w_nxt_pg,
    COUNT(DISTINCT CASE WHEN next_pageview_url = '/the-original-mr-fuzzy' THEN website_session_id ELSE NULL END) AS to_mrfuzzy,
    COUNT(DISTINCT CASE WHEN next_pageview_url = '/the-original-mr-fuzzy' THEN website_session_id ELSE NULL END)/COUNT(DISTINCT website_session_id) AS pct_to_mrfuzzy,
    COUNT(DISTINCT CASE WHEN next_pageview_url = '/the-forever-love-bear' THEN website_session_id ELSE NULL END) AS to_lovebear,
    COUNT(DISTINCT CASE WHEN next_pageview_url = '/the-forever-love-bear' THEN website_session_id ELSE NULL END)/COUNT(DISTINCT website_session_id) AS pct_to_lovebear
FROM next_pageviews
GROUP BY 1
ORDER BY 1 DESC;
```

11:94

Result Grid | Filter Rows: Search | Export:

time_line	product_sessio...	w_next_pg	pct_w_nxt_...	to_mrfuz...	pct_to_mrfu...	to_lovebear	pct_to_lovebe...
pre_product_launch	15696	11347	0.7229	11347	0.7229	0	0.0000
post_product_launch	10709	8200	0.7657	6654	0.6213	1546	0.1444

Findings:

- Previously, 72% of those who viewed the product showcase clicked on Mr. Fuzzy. Now, only 62% are engaging with Mr. Fuzzy, indicating a decrease. Additionally, there were no clicks on the love bear page initially, but after its launch, 14% of people clicked on it.

Product Level Conversion Funnel | SQL

```sql
-- Product Level Conversion Funnels
-- Since Jan 6th 2014
-- to April 14th 2014
-- Funnel products<product<to_cart<to_shipping<to_billing<to_thankyou
-- Step 1 Flaging the URLS with 1,0
SELECT
    website_session_id,
    pageview_url,
    created_at,
    CASE WHEN pageview_url = '/the-original-mr-fuzzy' THEN 1 ELSE 0 END AS fuzzy,
    CASE WHEN pageview_url = '/the-forever-love-bear' THEN 1 ELSE 0 END AS lovebear,
    CASE WHEN pageview_url = '/cart' THEN 1 ELSE 0 END AS cart,
    CASE WHEN pageview_url = '/shipping' THEN 1 ELSE 0 END AS shipping,
    CASE WHEN pageview_url = '/billing-2' THEN 1 ELSE 0 END AS billing,
    CASE WHEN pageview_url = '/thank-you-for-your-order' THEN 1 ELSE 0 END AS thankyou
FROM website_pageviews
WHERE created_at > '2013-01-06'
AND created_at < '2013-04-10'
AND pageview_url IN ('/the-original-mr-fuzzy', '/the-forever-love-bear', '/cart', '/shipping', '/billi
```

138:117

Result Grid | Filter Rows: Search | Export: | Fetch rows:

website_session...	pageview_url	created_at	fuzzy	lovebear	cart	shipping	billing	thankyou
63513	/shipping	2013-01-06 00:06:35	0	0	0	1	0	0
63513	/billing-2	2013-01-06 00:08:54	0	0	0	0	1	0
63513	/thank-you-for-your-order	2013-01-06 00:12:15	0	0	0	0	0	1
63515	/the-original-mr-fuzzy	2013-01-06 00:34:34	1	0	0	0	0	0
63515	/cart	2013-01-06 00:38:20	0	0	1	0	0	0
63516	/the-original-mr-fuzzy	2013-01-06 00:45:05	1	0	0	0	0	0
63517	/the-original-mr-fuzzy	2013-01-06 00:56:00	1	0	0	0	0	0

```sql
-- Step 2 Building the product funnel based on the flagging table
CREATE TEMPORARY TABLE product_funnel
SELECT
    website_session_id,
    MAX(fuzzy) AS flag_fuzzy,
    MAX(lovebear) AS flag_lovebear,
    MAX(cart) AS flag_cart,
    MAX(shipping) AS flag_shipping,
    MAX(billing) AS flag_billing,
    MAX(thankyou) AS flag_thankyou
FROM ( -- Step 1 SQL Code in subquery
GROUP BY 1;
SELECT * FROM product_funnel;
```

30:146

Result Grid | Filter Rows: Search | Export: | Fetch rows:

website_session...	flag_fuz...	flag_lovebear	flag_cart	flag_shippi...	flag_billi...	flag_thanky...
63516	1	0	0	0	0	0
63517	1	0	0	0	0	0
63518	1	0	1	0	0	0
63519	1	0	0	0	0	0
63520	1	0	0	0	0	0
63521	1	0	1	1	1	1
63526	1	0	0	0	0	0

Findings:

- The click-through rates on the product pages show some differences.
- About 43% of people who view Mr. Fuzzy click through to the cart, while almost 55% click through from the love bear page to the cart.
- However, once they reach the cart, the conversion rates are similar for both the products.

```sql
-- Step 3 Number Analysis
SELECT
    CASE WHEN flag_fuzzy = 1 THEN 'fuzzy_sessions'
         WHEN flag_lovebear = 1 THEN 'love_bear' ELSE 'check_logic' END AS prodcut_seen,
    COUNT(DISTINCT website_session_id) AS sessions,
    SUM(flag_cart) AS to_cart,
    SUM(flag_shipping) AS to_shipping,
    SUM(flag_billing) AS to_billing,
    SUM(flag_thankyou) AS to_thankyou
FROM product_funnel
GROUP BY 1;
```

12:158

Result Grid | Filter Rows: Search | Export:

prodcut_seen	sessions	to_cart	to_shipping	to_billing	to_thankyou
fuzzy_sessions	6985	3038	2084	1710	1088
love_bear	1599	877	603	488	301

```sql
-- Step 4 % Analysis
SELECT
    CASE WHEN flag_fuzzy = 1 THEN 'fuzzy_sessions'
         WHEN flag_lovebear = 1 THEN 'love_bear' ELSE 'check_logic' END AS prodcut_seen,
    COUNT(DISTINCT website_session_id) AS sessions,
    SUM(flag_cart)/COUNT(DISTINCT website_session_id) AS product_clickrate,
    SUM(flag_shipping)/SUM(flag_cart) AS cart_clickrate,
    SUM(flag_billing)/SUM(flag_shipping) AS shipping_clickrate,
    SUM(flag_thankyou)/SUM(flag_billing) AS billing_clickrate
FROM product_funnel
GROUP BY 1;
```

12:170

Result Grid | Filter Rows: Search | Export:

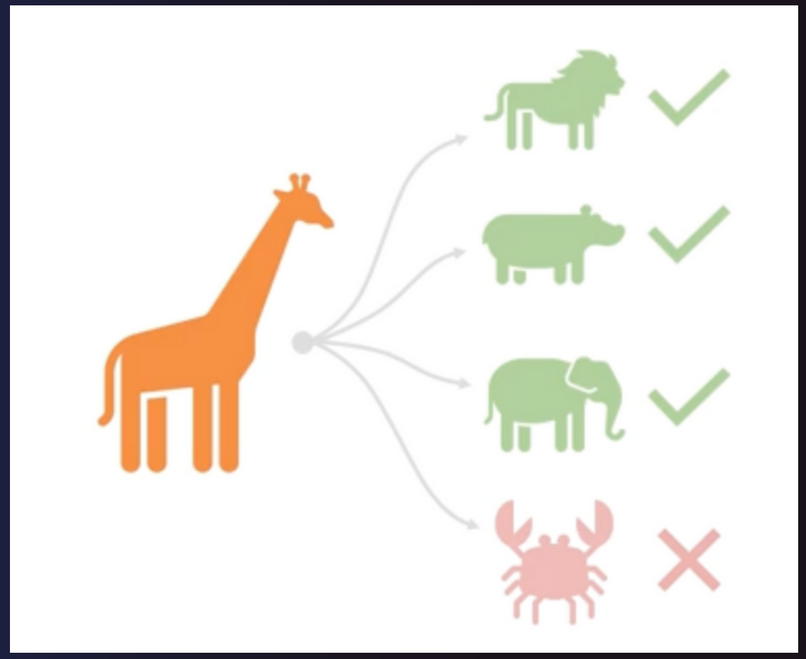
prodcut_seen	sessions	product_clickr...	cart_clickr...	shipping_clickr...	billing_clickr...
fuzzy_sessions	6985	0.4349	0.6860	0.8205	0.6363
love_bear	1599	0.5485	0.6876	0.8093	0.6168

Cross Selling Products

Cross Sell analysis is all about understanding which products users are most likely to purchase together and then making intelligent decisions about offering product recommendations so that you maximize revenue.

Some everyday use cases here include:

- Understanding which products are often purchased together, testing, and optimizing how we cross-sell products.
- Understanding the conversion rate impact and the overall revenue impact we see when trying to cross-sell additional products.



Cross Sell Analysis | SQL

```sql
-- On Spet 25, 2013 users get the option to add 2nd product while on cart page\
-- Comparing the following metrics month before and after the change
-- Click through rate of cartpage
-- Average products per order
-- Average order value
-- Revenue per cart page view
-- Step 1 Identifying relvant sessions
CREATE TEMPORARY TABLE cart_sessions
SELECT
    CASE WHEN created_at < '2013-09-25' THEN 'pre_cross_sell'
         WHEN created_at >= '2013-09-25' THEN 'post_cross_sell'
         ELSE 'check_logic' END AS time_period,
    website_session_id,
    website_pageview_id
FROM website_pageviews
WHERE created_at BETWEEN '2013-08-25' AND '2013-10-25'
AND pageview_url = '/cart';
SELECT * FROM cart_sessions;
```

29:189

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_period	website_session...	website_pageview...
pre_cross_sell	122947	285248
pre_cross_sell	122952	285261
pre_cross_sell	122954	285269
pre_cross_sell	122963	285290
pre_cross_sell	122973	285308
pre_cross_sell	122974	285315
pre_cross_sell	122987	285340
pre_cross_sell	123008	285376

```sql
-- Step 2
-- Now finding the website_sessions which clicked through the cart sessions
CREATE TEMPORARY TABLE after_cart_sessions
SELECT
    C.time_period,
    C.website_session_id,
    MIN(W.website_pageview_id) AS next_session_pageview_id
FROM cart_sessions AS C
    LEFT JOIN website_pageviews AS W
    ON C.website_session_id = W.website_session_id
    AND C.website_pageview_id < W.website_pageview_id
    -- This condintions help us only include the next pageview_id after cart session
GROUP BY 1, 2
HAVING MIN(W.website_pageview_id) IS NOT NULL;
-- Having condition limits the table to sessions which didn't bounce at the cart
SELECT * FROM after_cart_sessions;
```

35:206

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_period	website_session...	next_session_pageview...
pre_cross_sell	122947	285250
pre_cross_sell	122952	285262
pre_cross_sell	122954	285270
pre_cross_sell	122987	285342
pre_cross_sell	123008	285380
pre_cross_sell	123020	285404
pre_cross_sell	123021	285410
pre_cross_sell	123030	285429

```sql
-- Step 3
-- Now lets look at the orders which were successful after going through cart
-- Cart Orders which resulted in a sale
CREATE TEMPORARY TABLE cart_orders
SELECT
    C.time_period,
    C.website_session_id,
    O.order_id,
    O.items_purchased,
    O.price_usd
FROM cart_sessions AS C
        INNER JOIN orders AS O
        ON C.website_session_id = O.website_session_id;
SELECT * FROM cart_orders;
```

27:221

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_period	website_session...	order_id	items_purchas...	price_usd
pre_cross_sell	122947	6645	1	49.99
pre_cross_sell	122987	6646	1	49.99
pre_cross_sell	123008	6647	1	49.99
pre_cross_sell	123021	6648	1	59.99
pre_cross_sell	123030	6649	1	49.99
pre_cross_sell	123031	6650	1	49.99
pre_cross_sell	123038	6651	1	49.99
pre_cross_sell	123049	6652	1	49.99
pre_cross_sell	123055	6653	1	49.99

```sql
-- Step 4 Bringing All together
SELECT
    C.time_period,
    C.website_session_id,
    CASE WHEN A.next_session_pageview_id IS NULL THEN 0 ELSE 1 END AS flag_anotherpageview,
    CASE WHEN O.order_id IS NULL THEN 0 ELSE 1 END AS placed_order,
    O.items_purchased,
    O.price_usd
FROM cart_sessions AS C
    LEFT JOIN after_cart_sessions AS A
        ON C.website_session_id = A.website_session_id
    LEFT JOIN cart_orders AS O
        ON C.website_session_id = O.website_session_id
ORDER BY 2;
```

38:233

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_period	website_session...	flag_anotherpagevi...	placed_order	items_purchas...	price_usd
pre_cross_sell	122954	1	0	NULL	NULL
pre_cross_sell	122963	0	0	NULL	NULL
pre_cross_sell	122973	0	0	NULL	NULL
pre_cross_sell	122974	0	0	NULL	NULL
pre_cross_sell	122987	1	1	1	49.99
pre_cross_sell	123008	1	1	1	49.99
pre_cross_sell	123016	0	0	NULL	NULL
pre_cross_sell	123020	1	0	NULL	NULL

```sql
-- Step 5
SELECT
    time_period,
    COUNT(website_session_id) AS cart_sessions,
    SUM(flag_anotherpageview) AS clicked_next_page,
    SUM(flag_anotherpageview)/COUNT(website_session_id) AS cart_click_rate,
    SUM(placed_order) AS orders_placed,
    SUM(items_purchased) AS products_purchased,
    SUM(items_purchased)/SUM(placed_order) AS products_per_order,
    SUM(price_usd)/SUM(placed_order) AS aov,
    SUM(price_usd)/COUNT(website_session_id) AS revenue_per_cartpage_view
FROM ( -- Step 4 as Subquery
GROUP BY 1;
```

12:261

Result Grid | Filter Rows: Search | Export:

time_period	cart_sessions	clicked_next_pa...	cart_click_r...	orders_plac...	products_purchas...	products_per_or...	aov	revenue_per_cartpage_vi...
pre_cross_sell	1830	1229	0.6716	652	652	1.0000	51.416380	18.318842
post_cross_sell	1975	1351	0.6841	671	701	1.0447	54.251848	18.431894

Impact of New Product Launch | SQL

```sql
-- Portfolio Expansion Analysis
-- Birthday Bear launched on December 12th, 2013
-- Pre and post analysis comparing
-- month before v/s month after
-- session to order conversion rate
-- Average Order Value
-- Products per order and revenue per session
-- Step 1 Bringing all the relevant data into one table
SELECT
    CASE WHEN W.created_at < '2013-12-12' THEN 'pre_launch'
         WHEN W.created_at >= '2013-12-12' THEN 'post_launch' ELSE 'check_logic' END AS time_period,
    W.website_session_id,
    O.order_id,
    O.items_purchased,
    O.price_usd
FROM website_sessions AS W
    LEFT JOIN orders AS O
    ON W.website_session_id = O.website_session_id
WHERE W.created_at BETWEEN '2013-11-12' AND '2014-01-12';
```

27:279

Result Grid | Filter Rows: Search | Export: | Fetch rows:

time_period	website_session_id	order_id	items_purchas...	price_usd
pre_launch	149033	8342	1	49.99
pre_launch	149034	NULL	NULL	NULL
pre_launch	149035	NULL	NULL	NULL
pre_launch	149036	NULL	NULL	NULL
pre_launch	149037	NULL	NULL	NULL
pre_launch	149038	NULL	NULL	NULL

```sql
-- Step 2 getting the numbers
SELECT
    time_period,
    COUNT(website_session_id) AS sessions,
    COUNT(order_id) AS orders,
    COUNT(order_id)/COUNT(website_session_id) AS conversion_rate,
    SUM(price_usd) AS revenue,
    SUM(price_usd)/COUNT(order_id) AS AOV,
    SUM(items_purchased) AS products_sold,
    SUM(items_purchased)/COUNT(order_id) AS products_per_order,
    SUM(price_usd)/COUNT(website_session_id) AS revenue_per_session
FROM(    -- Step 1 as subquery
GROUP BY 1;
```

12:307

Result Grid | Filter Rows: Search | Export:

time_period	sessions	orders	conversion_rate	revenue	AOV	products_s...	products_per_or...	revenue_per_sessi...
pre_launch	17343	1055	0.0608	57208.96	54.226502	1104	1.0464	3.298677
post_launch	13383	940	0.0702	53515.44	56.931319	1056	1.1234	3.998763

Findings:

- The introduction of the Birthday Bear has resulted in an improved conversion rate, increasing from 6% to 7%.
- The average order value has also shown an increase, rising from $54.22 to $56.93.
- The number of products per order has experienced a slight increase, going from 1.04 to 1.12. This suggests that the Birthday Bear may serve as an effective cross-seller for Product One and Product Two. It is possible that the Birthday Bear sells well as a primary product, while Product One and Product Two perform well as complementary items in conjunction with the bear.
- Furthermore, the revenue per session has significantly improved from $3.29 to $3.99. This signifies a notable increase in revenue generated from each browsing session.

Product Refund Analysis

Analyzing product refund rates is all about controlling for quality and understanding where you might have problems to address.

Some everyday use cases here include:

- Monitoring products from different suppliers
- Understanding refund rates for products at various price points
- Finally, we want to take product refund rates into account when we're looking at revenue and the overall performance of our business.

Product Refund Analysis | SQL

```sql
-- Product Refund Rates of Fuzzy
-- Step 1
SELECT
    O.created_at,
    O.product_id,
    O.order_item_id,
    CASE WHEN R.order_item_refund_id IS NULL THEN 0 ELSE 1 END AS refund_status
FROM order_items AS O
            LEFT JOIN order_item_refunds AS R
            ON O.order_item_id = R.order_item_id
WHERE O.created_at < '2014-10-15';
```

35:337

Result Grid | Filter Rows: Search | Export: | Fetch rows:

created_at	product_id	order...	refund_stat...
2012-03-19 10:42:46	1	1	0
2012-03-19 19:27:37	1	2	0
2012-03-20 06:44:45	1	3	0
2012-03-20 09:41:45	1	4	0
2012-03-20 11:28:15	1	5	0
2012-03-20 16:12:47	1	6	0
2012-03-20 17:03:41	1	7	0
2012-03-20 23:35:27	1	8	0
2012-03-21 02:35:01	1	9	0

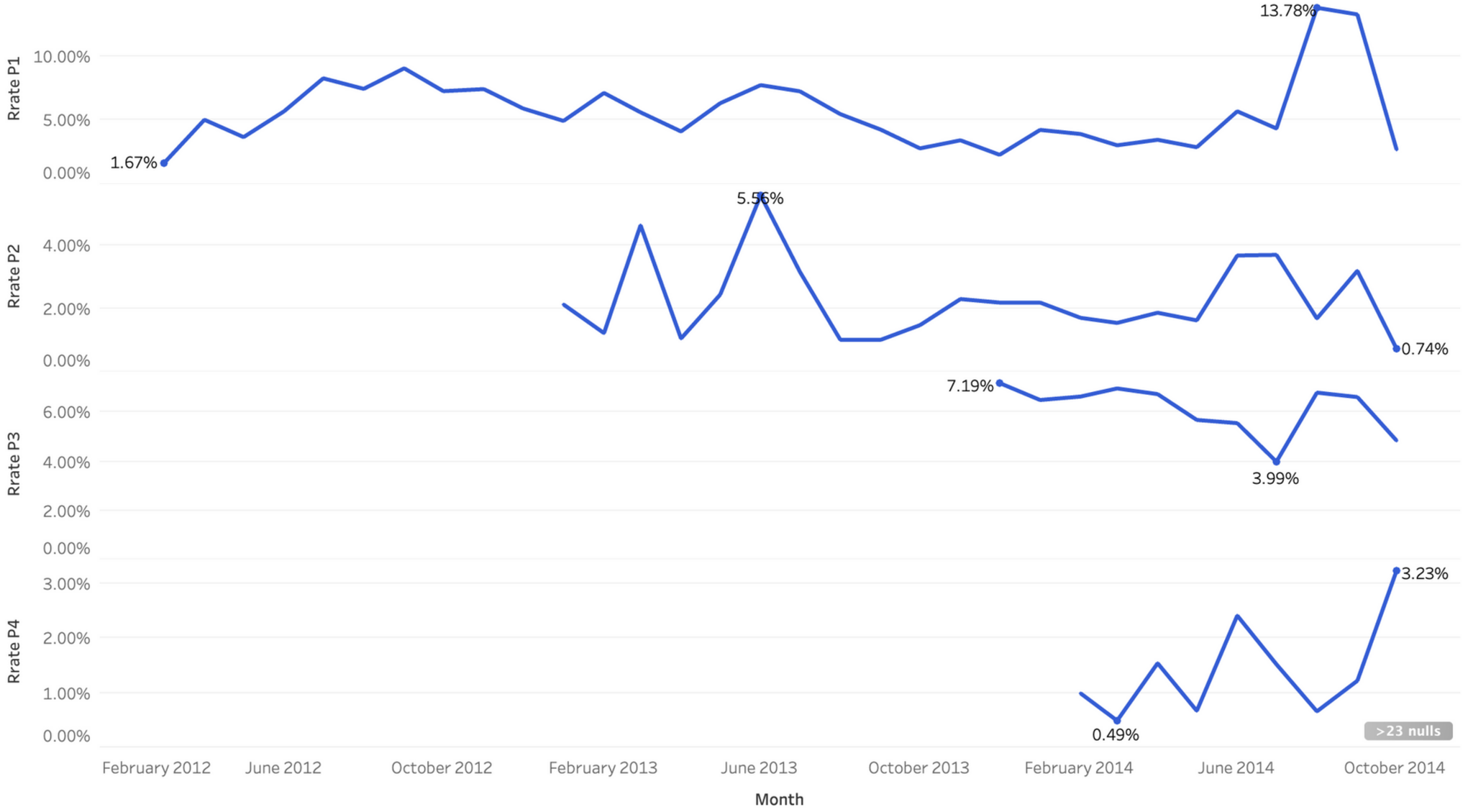
```sql
SELECT
    YEAR(created_at),
    MONTH(created_at),
    COUNT(CASE WHEN product_id =1 THEN order_item_id ELSE NULL END) AS orders_p1,
    SUM(CASE WHEN product_id =1 AND refund_status =1 THEN 1 ELSE 0 END)/COUNT(CASE WHEN product_id =1 THEN order_item_id ELSE NULL END) AS rrate_p1,
    COUNT(CASE WHEN product_id =2 THEN order_item_id ELSE NULL END) AS orders_p2,
    SUM(CASE WHEN product_id =2 AND refund_status =1 THEN 1 ELSE 0 END)/COUNT(CASE WHEN product_id =2 THEN order_item_id ELSE NULL END) AS rrate_p2,
    COUNT(CASE WHEN product_id =3 THEN order_item_id ELSE NULL END) AS orders_p3,
    SUM(CASE WHEN product_id =3 AND refund_status =1 THEN 1 ELSE 0 END)/COUNT(CASE WHEN product_id =3 THEN order_item_id ELSE NULL END) AS rrate_p3,
    COUNT(CASE WHEN product_id =4 THEN order_item_id ELSE NULL END) AS orders_p4,
    SUM(CASE WHEN product_id =4 AND refund_status =1 THEN 1 ELSE 0 END)/COUNT(CASE WHEN product_id =4 THEN order_item_id ELSE NULL END) AS rrate_p4
FROM ( -- Step 1 as Sub Query
  GROUP BY 1,2 ;
```

30:350

Result Grid | Filter Rows: Search | Export:

YEAR(created_at)	MONTH(created_...	orders_p1	rrate_p1	orders_p2	rrate_p2	orders_p3	rrate_p3	orders_p4	rrate_p4
2012	3	60	0.0167	0	NULL	0	NULL	0	NULL
2012	4	99	0.0505	0	NULL	0	NULL	0	NULL
2012	5	108	0.0370	0	NULL	0	NULL	0	NULL
2012	6	140	0.0571	0	NULL	0	NULL	0	NULL
2012	7	169	0.0828	0	NULL	0	NULL	0	NULL
2012	8	228	0.0746	0	NULL	0	NULL	0	NULL
2012	9	287	0.0906	0	NULL	0	NULL	0	NULL
2012	10	371	0.0728	0	NULL	0	NULL	0	NULL
2012	11	618	0.0744	0	NULL	0	NULL	0	NULL
2012	12	506	0.0593	0	NULL	0	NULL	0	NULL
2013	1	343	0.0496	47	0.0213	0	NULL	0	NULL
2013	2	336	0.0714	162	0.0123	0	NULL	0	NULL
2013	3	320	0.0563	65	0.0462	0	NULL	0	NULL

Product Refund Insights: Monthly Refund Percentage Analysis



Product Refund Analysis

Analyzing product refund rates is all about controlling for quality and understanding where you might have problems to address.

Some everyday use cases here include:

- Monitoring products from different suppliers
- Understanding refund rates for products at various price points
- Finally, we want to take product refund rates into account when we're looking at revenue and the overall performance of our business.