estudo_pix_maquininha (Python)

Import notebook

ID PIX MAQUININHA (int_dados_negocios.id_pix_maquininha)

- cliente_id: Identificador único do cliente
- segmento: Categoria do lojista: micro, pequeno, médio
- uf: Estado de atuação (ex: RJ, SP, MG, ...)
- tempo_cliente_meses: Tempo (em meses) desde ativação da conta
- tpv_total_medio: Volume médio mensal total (todas as formas de pagamento) --> então aqui vai entrar não apenas a forma de pagamento via pix maquininha
- saldo_medio_conta: Saldo médio mantido na conta da empresa

VENDAS PIX MAQUININHA (int_dados_negocios.vendas_pix_maquininha)

- cliente_id: Identificador do cliente (chave para clientes)
- mes: Referência de mês no formato YYYY-MM (ex: 2024-06)
- tpv_pix: Volume transacionado via Pix na maquininha naquele mês -> aqui define quem usa e quem não usa pix na maquininha
- receita_pix: Receita gerada para a empresa neste mês via Pix
- saldo_conta: Saldo médio mantido na conta neste mês
- grupo_experimento: "controle" ou "tratamento" — alocação no experimento
- participou_promocao: 1 se o cliente recebeu a promoção no mês de referência, 0 caso contrário

ETL Inicial

Setando o Schema e Referências das Tabelas

```
CATALOGO = "int_dados_negocios"
TBL_CLIENTES = f"{CATALOGO}.id_pix_maquininha"
TBL_VENDAS = f"{CATALOGO}.vendas_pix_maquininha"

spark.sql(f"USE {CATALOGO}")
```

```
DataFrame[]
```

Criando as TEMP VIEW em SQL

;

```
%sql

-- tabela clientes id
CREATE OR REPLACE TEMP VIEW vw_clientes AS
SELECT
  cliente_id,
  segmento,
  uf,
  tempo_cliente_meses,
  tpv_total_medio,
  saldo_medio_conta
FROM id_pix_maquininha;

-- tabela de vendas
CREATE OR REPLACE TEMP VIEW vw_vendas_pix AS
SELECT
  cliente_id,
  mes,
  tpv_pix,
  receita_pix,
  saldo_conta,
  grupo_experimento
FROM vendas_pix_maquininha
```

Join entre as tabelas

```
%sql
CREATE OR REPLACE TEMP VIEW vw_base_mensal AS
WITH base AS (
  SELECT
      v.mes,
      v.cliente_id,
      c.segmento,
      c.uf,
      c.tempo_cliente_meses,
      c.tpv_total_medio,
      c.saldo_medio_conta,
      v.tpv_pix,
      v.receita_pix,
      v.saldo_conta,
      v.grupo_experimento
  FROM vw_vendas_pix v
  LEFT JOIN vw_clientes c
    ON v.cliente_id = c.cliente_id
)
SELECT
    mes,
    cliente_id,
    segmento,
    uf,
    tempo_cliente_meses,
    tpv_total_medio,
    saldo_medio_conta,
    tpv_pix,
    receita_pix,
    saldo_conta,
    grupo_experimento,

    -- Flag por cliente: soma total de TPV Pix > 0 em todos os períodos
    CASE
      WHEN SUM(COALESCE(tpv_pix, 0)) OVER (PARTITION BY cliente_id) > 0 THEN 1
      ELSE 0
    END AS usuario_pix,

    -- Participação em promoção: 1 se grupo_experimento não é NULL, senão 0
    CASE
      WHEN grupo_experimento IS NULL THEN 0
      ELSE 1
    END AS participou_promocao
FROM base
```

Tabela temporária sem duplicatas

```
%sql
CREATE OR REPLACE TEMP VIEW vw_perfil_usuarios AS
SELECT DISTINCT
    cliente_id,
    segmento,
    uf,
    tempo_cliente_meses,
    tpv_total_medio,
    saldo_medio_conta,
    usuario_pix
FROM vw_base_mensal
```

1ª Etapa: Análise Exploratória

1.1 Descreva o perfil dos clientes que usam o produto Pix na maquininha versus os que não usam.

Usabilidade de PIX na maquininha por segmento

```
%sql
SELECT
    segmento,
    SUM(CASE WHEN usuario_pix = 1 THEN 1 ELSE 0 END) AS usuarios,
    SUM(CASE WHEN usuario_pix = 0 THEN 1 ELSE 0 END) AS nao_usuarios,
    COUNT(*) AS total,
    ROUND(
        SUM(CASE WHEN usuario_pix = 1 THEN 1 ELSE 0 END) / NULLIF(COUNT(*), 0),
        4
    ) AS pct_usuarios
FROM vw_perfil_usuarios
GROUP BY segmento
ORDER BY usuarios DESC;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [segmento: string, usuarios: long ... 3 more fields]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
%sql

WITH classificados AS (
  SELECT
    segmento,
    CASE WHEN COALESCE(tpv_pix, 0) > 0 THEN 'Usuário' ELSE 'Não usuário' END AS status_uso,
    cliente_id
  FROM vw_base_mensal
),

contagem AS (
  SELECT
    segmento,
    status_uso,
    COUNT(DISTINCT cliente_id) AS clientes
  FROM classificados
  GROUP BY segmento, status_uso
),

totais AS (
  SELECT
    segmento,
    SUM(clientes) AS total_segmento
  FROM contagem
  GROUP BY segmento
)

SELECT
  c.segmento,
  c.status_uso,
  c.clientes,
  t.total_segmento,
  ROUND(100.0 * c.clientes / t.total_segmento, 2) AS pct_clientes
FROM contagem c
JOIN totais t
  ON c.segmento = t.segmento
ORDER BY c.segmento, c.status_uso
```

_sqldf: pyspark.sql.dataframe.DataFrame = [segmento: string, status_uso: string ... 3 more fields]

Table **usuarios_e_nao_usuarios**

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análise tempo de casa

```
%sql
WITH base AS (
    SELECT
        tempo_cliente_meses,
        usuario_pix,
        CASE
            WHEN tempo_cliente_meses IS NULL THEN 'Sem informação'
            WHEN tempo_cliente_meses BETWEEN 0 AND 3  THEN '0 a 3'
            WHEN tempo_cliente_meses BETWEEN 4 AND 6  THEN '4 a 6'
            WHEN tempo_cliente_meses BETWEEN 7 AND 9  THEN '7 a 9'
            WHEN tempo_cliente_meses BETWEEN 10 AND 12 THEN '10 a 12'
            WHEN tempo_cliente_meses BETWEEN 13 AND 24 THEN '13 a 24'
            WHEN tempo_cliente_meses BETWEEN 25 AND 36 THEN '25 a 36'
            WHEN tempo_cliente_meses BETWEEN 37 AND 48 THEN '37 a 48'
            WHEN tempo_cliente_meses >= 49 THEN '49+'
            ELSE 'Outros'
        END AS faixa_tempo
    FROM vw_perfil_usuarios
)
SELECT
    faixa_tempo,
    -- proporções em decimal (0–1)
    100 * ROUND(
        SUM(CASE WHEN usuario_pix = 1 THEN 1 ELSE 0 END) / NULLIF(COUNT(*), 0),
        4
    ) AS prop_usuarios,
    100 * ROUND(
        SUM(CASE WHEN usuario_pix = 0 THEN 1 ELSE 0 END) / NULLIF(COUNT(*), 0),
        4
    ) AS prop_nao_usuarios
FROM base
GROUP BY faixa_tempo
ORDER BY CASE faixa_tempo
    WHEN '0 a 3'    THEN 1
    WHEN '4 a 6'    THEN 2
    WHEN '7 a 9'    THEN 3
    WHEN '10 a 12'  THEN 4
    WHEN '13 a 24'  THEN 5
    WHEN '25 a 36'  THEN 6
    WHEN '37 a 48'  THEN 7
    WHEN '49+'      THEN 8
    WHEN 'Sem informação' THEN 9
    ELSE 99
END
```

_sqldf: pyspark.sql.dataframe.DataFrame = [faixa_tempo: string, prop_usuarios: double ... 1 more field]

Table **Visualization 1**

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análise por UF

```sql
%sql
CREATE OR REPLACE TEMP VIEW vw_resumo_uf AS
WITH dim_uf AS (
  SELECT * FROM VALUES
    ('AC','Norte','Rio Branco'),
    ('AL','Nordeste','Maceió'),
    ('AP','Norte','Macapá'),
    ('AM','Norte','Manaus'),
    ('BA','Nordeste','Salvador'),
    ('CE','Nordeste','Fortaleza'),
    ('DF','Centro-Oeste','Brasília'),
    ('ES','Sudeste','Vitória'),
    ('GO','Centro-Oeste','Goiânia'),
    ('MA','Nordeste','São Luís'),
    ('MT','Centro-Oeste','Cuiabá'),
    ('MS','Centro-Oeste','Campo Grande'),
    ('MG','Sudeste','Belo Horizonte'),
    ('PA','Norte','Belém'),
    ('PB','Nordeste','João Pessoa'),
    ('PR','Sul','Curitiba'),
    ('PE','Nordeste','Recife'),
    ('PI','Nordeste','Teresina'),
    ('RJ','Sudeste','Rio de Janeiro'),
    ('RN','Nordeste','Natal'),
    ('RS','Sul','Porto Alegre'),
    ('RO','Norte','Porto Velho'),
    ('RR','Norte','Boa Vista'),
    ('SC','Sul','Florianópolis'),
    ('SE','Nordeste','Aracaju'),
    ('SP','Sudeste','São Paulo'),
    ('TO','Norte','Palmas')
  AS d(uf, regiao, capital)
)
SELECT
  d.uf,
  d.regiao,
  d.capital,
  COUNT(*) AS total_base,
  SUM(CASE WHEN v.usuario_pix = 1 THEN 1 ELSE 0 END) AS qtd_usuarios,
  SUM(CASE WHEN v.usuario_pix = 0 THEN 1 ELSE 0 END) AS qtd_nao_usuarios
FROM vw_perfil_usuarios v
LEFT JOIN dim_uf d
  ON v.uf = d.uf
GROUP BY d.uf, d.regiao, d.capital
```

```sql
%sql
SELECT *
FROM vw_resumo_uf
```

_sqldf: pyspark.sql.dataframe.DataFrame = [uf: string, regiao: string ... 4 more fields]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Por regional

```
%sql
SELECT
    regiao,
    SUM(total_base) AS total_base,
    ROUND(SUM(qtd_usuarios) / NULLIF(SUM(total_base), 0), 4) AS prop_usuarios,
    ROUND(SUM(qtd_nao_usuarios) / NULLIF(SUM(total_base), 0), 4) AS prop_nao_usuarios
FROM vw_resumo_uf
GROUP BY regiao
ORDER BY total_base DESC
```

_sqldf: pyspark.sql.dataframe.DataFrame = [regiao: string, total_base: long ... 2 more fields]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
%sql
SELECT
    uf,
    total_base,
    ROUND(qtd_usuarios / NULLIF(total_base, 0), 4) AS prop_usuarios,
    ROUND(qtd_nao_usuarios / NULLIF(total_base, 0), 4) AS prop_nao_usuarios
FROM vw_resumo_uf
ORDER BY total_base DESC
```

_sqldf: pyspark.sql.dataframe.DataFrame = [uf: string, total_base: long ... 2 more fields]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

1.2 Analise as tendências de uso do produto ao longo dos meses no uso do Pix (TPV médio, churn, receita, etc).

Inclusão da coluna aceitou_pgto_pix

Indica se o cliente aceitou pagamento pix no mês de referência

```
%sql
CREATE OR REPLACE TEMP VIEW vw_base_mensal_enriquecida AS
SELECT
  mes,
  cliente_id,
  segmento,
  uf,
  tempo_cliente_meses,
  tpv_total_medio,
  saldo_medio_conta,
  tpv_pix,
  receita_pix,
  saldo_conta,
  grupo_experimento,
  usuario_pix,
  participou_promocao,
  CASE WHEN COALESCE(tpv_pix, 0) > 0 THEN 1 ELSE 0 END AS aceitou_pgto_pix
  FROM vw_base_mensal
```

TPV, RECEITA, QTD USUÁRIOS TOTAIS

```
%sql
SELECT *
FROM vw_base_mensal_enriquecida
WHEN
```

_sqldf: pyspark.sql.dataframe.DataFrame = [mes: date, cliente_id: long ... 12 more fields]

Table receita por segmento

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análises por segmento e mês

```
%sql
SELECT
  mes,
  segmento,
  AVG(tpv_pix) AS tpv_medio_pix,
  AVG(saldo_conta) AS saldo_medio_mes,
  100 * AVG(
    CASE
      WHEN tpv_pix > 0 AND receita_pix IS NOT NULL THEN receita_pix / tpv_pix
      ELSE NULL
    END
  ) AS taxa_media_utilizacao_pct,
  COUNT(*) AS qtd_clientes
FROM vw_base_mensal_enriquecida
WHERE aceitou_pgto_pix = 1
GROUP BY mes, segmento
ORDER BY mes, segmento;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [mes: date, segmento: string ... 4 more fields]

Table averages tpv por segmento taxa media segmento

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análises entre usuários e não usuários

```
%sql
SELECT
  CASE
    WHEN usuario_pix = 1 THEN 'usuário'
    ELSE 'não usuário'
  END AS tipo_pj_pix,
  COUNT(*) AS qtd_clientes,
  AVG(COALESCE(saldo_medio_conta, 0)) AS saldo_medio_conta_medio,
  AVG(COALESCE(tpv_total_medio, 0))   AS tpv_total_medio_medio
FROM vw_perfil_usuarios
GROUP BY
  CASE
    WHEN usuario_pix = 1 THEN 'usuário'
    ELSE 'não usuário'
  END
ORDER BY tipo_pj_pix DESC
```

_sqldf: pyspark.sql.dataframe.DataFrame = [tipo_pj_pix: string, qtd_clientes: long ... 2 more fields]

Table **barras comparando**

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análises por quantidade de utilização

```
%sql
WITH por_cliente AS (
  SELECT
    cliente_id,
    SUM(aceitou_pgto_pix) AS aceitou_pgto_pix_sum,
    AVG(COALESCE(tpv_pix, 0)) AS tpv_pix_medio,
    AVG(COALESCE(saldo_conta, 0)) AS saldo_conta_medio
  FROM vw_base_mensal_enriquecida
  WHERE usuario_pix = 1
  GROUP BY cliente_id
),
com_faixa AS (
  SELECT
    cliente_id,
    aceitou_pgto_pix_sum,
    tpv_pix_medio,
    saldo_conta_medio,
    CASE
      WHEN aceitou_pgto_pix_sum BETWEEN 1 AND 3  THEN '1-3'
      WHEN aceitou_pgto_pix_sum BETWEEN 4 AND 6  THEN '4-6'
      WHEN aceitou_pgto_pix_sum BETWEEN 7 AND 9  THEN '7-9'
      WHEN aceitou_pgto_pix_sum BETWEEN 10 AND 12 THEN '10-12'
      ELSE 'fora_da_faixa'
    END AS faixa_aceitou_pix
  FROM por_cliente
)
SELECT
  faixa_aceitou_pix,
  COUNT(*)                        AS qtd_clientes,
  AVG(tpv_pix_medio)              AS tpv_pix_medio,
  AVG(saldo_conta_medio)          AS saldo_conta_medio
FROM com_faixa
WHERE faixa_aceitou_pix <> 'fora_da_faixa'
GROUP BY faixa_aceitou_pix
ORDER BY
  CASE faixa_aceitou_pix
    WHEN '1-3'   THEN 1
    WHEN '4-6'   THEN 2
    WHEN '7-9'   THEN 3
    WHEN '10-12' THEN 4
    ELSE 5
  END;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [faixa_aceitou_pix: string, qtd_clientes: long ... 2 more fields]

Table Visualization 1

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análises entre usuários e não usuários do segmento micro

```
%sql

-- 1) Média por cliente e usuário_pix (apenas segmento micro)
WITH media_por_cliente AS (
  SELECT
    cliente_id,
    usuario_pix,
    AVG(COALESCE(saldo_medio_conta, 0)) AS saldo_medio_conta_avg_cliente
  FROM vw_base_mensal_enriquecida
  WHERE segmento = 'micro'
  GROUP BY cliente_id, usuario_pix
)

-- 2) Média por usuario_pix (a média das médias de cada cliente)
SELECT
  usuario_pix,
  AVG(saldo_medio_conta_avg_cliente) AS saldo_medio_conta_avg_usuario
FROM media_por_cliente
GROUP BY usuario_pix;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [usuario_pix: integer, saldo_medio_conta_avg_usuario: decimal(26,10)]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

CHURN

```
%sql
CREATE OR REPLACE TEMP VIEW vw_cohort_inicio AS
SELECT
  cliente_id,
  TRUNC(MIN(mes), 'MM') AS mes_inicio
FROM vw_base_mensal_enriquecida
WHERE COALESCE(tpv_pix, 0) > 0
GROUP BY cliente_id;
```

```
%sql

CREATE OR REPLACE TEMP VIEW vw_activity AS
SELECT
  b.cliente_id,
  TRUNC(b.mes, 'MM') AS mes,
  CASE WHEN COALESCE(b.tpv_pix, 0) > 0 THEN 1 ELSE 0 END AS ativo
FROM vw_base_mensal_enriquecida b;
```

```
%sql
CREATE OR REPLACE TEMP VIEW vw_cohort_stages AS
SELECT
  a.cliente_id,
  i.mes_inicio,
  a.mes AS mes_atividade,
  (YEAR(a.mes) - YEAR(i.mes_inicio)) * 12 + (MONTH(a.mes) - MONTH(i.mes_inicio)) AS stage
FROM vw_activity a
JOIN vw_cohort_inicio i USING (cliente_id)
WHERE a.ativo = 1
```

```
%sql

CREATE OR REPLACE TEMP VIEW vw_cohort_agg AS
WITH bucket AS (
  SELECT
    mes_inicio AS date,
    COUNT(DISTINCT cliente_id) AS bucket_population_size
  FROM vw_cohort_stages
  WHERE stage = 0
  GROUP BY mes_inicio
),
active AS (
  SELECT
    mes_inicio AS date,
    stage,
    COUNT(DISTINCT cliente_id) AS stage_value_count
  FROM vw_cohort_stages
  GROUP BY mes_inicio, stage
)
SELECT
  a.date,
  a.stage,
  b.bucket_population_size,
  a.stage_value_count,
  ROUND(a.stage_value_count * 1.0 / b.bucket_population_size, 4) AS stage_value_rate
FROM active a
JOIN bucket b USING (date)
ORDER BY a.date, a.stage;
```

```
%sql
select * from vw_cohort_agg
```

_sqldf: pyspark.sql.dataframe.DataFrame = [date: date, stage: integer ... 3 more fields]

Table Visualization 1

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

2ª Etapa: Avaliação da Promoção

Temp View apenas com os grupos de interesse

```
%sql
CREATE OR REPLACE TEMP VIEW vw_base_mensal_grupos_tratamento AS
SELECT
  mes,
  cliente_id,
  segmento,
  uf,
  tempo_cliente_meses,
  tpv_total_medio,
  saldo_medio_conta,
  tpv_pix,
  receita_pix,
  saldo_conta,
  grupo_experimento,
  usuario_pix,
  participou_promocao,
  aceitou_pgto_pix
FROM vw_base_mensal_enriquecida
WHERE grupo_experimento IS NOT NULL
```

Quantidade por grupo de tratamento

```
%sql
WITH base AS (
  SELECT *
  FROM vw_base_mensal_enriquecida
  WHERE grupo_experimento IS NOT NULL
),
min_por_cliente_grupo AS (
  SELECT
    cliente_id,
    grupo_experimento AS grupo_tratamento,
    MIN(CASE WHEN aceitou_pgto_pix > 0 THEN mes END) AS mes_min_aceite_pix
  FROM base
  GROUP BY cliente_id, grupo_experimento
)

SELECT
  grupo_tratamento,
  COUNT(mes_min_aceite_pix) AS quantidade,
  SUM(CASE WHEN mes_min_aceite_pix = DATE '2024-06-01' THEN 1 ELSE 0 END) AS qtd_iniciaram_free_trial -- aqui
eu quero entender quantos que iniciaram a utilização de pix na maquininha no mes da promoção
FROM min_por_cliente_grupo
GROUP BY grupo_tratamento
ORDER BY grupo_tratamento
```

› _sqldf: pyspark.sql.dataframe.DataFrame = [grupo_tratamento: string, quantidade: long ... 1 more field]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Informações que eu já tenho:

- Todos de ambos grupos começaram a utilizar a função pix na maquininha no mês de junho/24
- Como todos iniciaram em junho, e temos que o CHURN analisado anteriormente é igual a 0, isso quer dizer que todos continuaram utilizando

Gráficos simples para comparação

```
%sql
SELECT *
FROM vw_base_mensal_grupos_tratamento
```

_sqldf: pyspark.sql.dataframe.DataFrame = [mes: date, cliente_id: long ... 12 more fields]

Table soma de tpv_pix receita Visualization 1

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Análise estatística: diferença-em-diferenças

```
%sql

-- 2) Base de DiD (pré/pós) com indicadores treated e post
CREATE OR REPLACE TEMP VIEW did_base AS
SELECT
  mes,
  cliente_id,
  CAST(tpv_pix AS DOUBLE) AS tpv_pix,
  CASE WHEN grupo_experimento = 'tratamento' THEN 1 ELSE 0 END AS treated,
  CASE WHEN mes >= DATE '2024-06-01' THEN 1 ELSE 0 END AS post
FROM vw_base_mensal_grupos_tratamento
WHERE mes IN (DATE '2024-05-01', DATE '2024-06-01');
```

```
%sql
-- 3) Deduplicação cliente-mês
CREATE OR REPLACE TEMP VIEW did_base_clean AS
SELECT
  mes,
  cliente_id,
  MAX(treated) AS treated,
  MAX(post)    AS post,
  MAX(tpv_pix) AS tpv_pix
FROM did_base GROUP BY mes, cliente_id
```

```
%sql

-- 4) DiD clássico (SQL): média por grupo e período, deltas e diferença das deltas
WITH grp AS (
  SELECT
    treated,
    post,
    AVG(tpv_pix) AS avg_tpv
  FROM did_base_clean
  GROUP BY treated, post
),
deltas AS (
  SELECT
    treated,
    MAX(CASE WHEN post = 1 THEN avg_tpv END)
  - MAX(CASE WHEN post = 0 THEN avg_tpv END) AS delta
  FROM grp
  GROUP BY treated
)
SELECT
  MAX(CASE WHEN treated = 1 THEN delta END)
- MAX(CASE WHEN treated = 0 THEN delta END) AS did_tpv_pix
FROM deltas
```

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [did_tpv_pix: double]

Table

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
# 5) DiD via OLS com erro-padrão clusterizado por cliente

import pandas as pd
import statsmodels.formula.api as smf

# Puxar a temp view diretamente
df = spark.sql("""
SELECT
  mes,
  cliente_id,
  CAST(tpv_pix AS DOUBLE) AS tpv_pix,
  CASE WHEN grupo_experimento = 'tratamento' THEN 1 ELSE 0 END AS treated,
  CASE WHEN mes >= DATE '2024-06-01' THEN 1 ELSE 0 END AS post
FROM vw_base_mensal_grupos_tratamento
WHERE mes IN (DATE '2024-05-01', DATE '2024-06-01')
""")

pdf = df.toPandas()

# Criar interação do DiD
pdf["did"] = pdf["treated"] * pdf["post"]

# Regressão OLS com erro-padrão clusterizado por cliente
model = smf.ols("tpv_pix ~ treated + post + did", data=pdf).fit(
    cov_type="cluster",
    cov_kwds={"groups": pdf["cliente_id"]}
)

print(model.summary())
```

df: pyspark.sql.dataframe.DataFrame = [mes: date, cliente_id: long ... 3 more fields]

pdf: pandas.core.frame.DataFrame = [mes: object, cliente_id: int64 ... 4 more fields]

```
Time:                        04:15:37   Log-Likelihood:                -6131.8
No. Observations:                 566   AIC:                          1.227e+04
Df Residuals:                     562   BIC:                          1.229e+04
Df Model:                           3
Covariance Type:            cluster
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1.271e-11   3.66e-12      3.469      0.001    5.53e-12    1.99e-11
treated     6.275e-12   2.46e-12      2.551      0.011    1.45e-12    1.11e-11
post        9031.8408    971.815      9.294      0.000    7127.118    1.09e+04
did         4886.6853   2075.735      2.354      0.019     818.319    8955.052
==============================================================================
Omnibus:                      531.211   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            18593.084
Skew:                           4.138   Prob(JB):                         0.00
Kurtosis:                      29.831   Cond. No.                         6.84
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
```

3ª Etapa: Estratégias de Crescimento

1ª iniativa: Isenção de taxa de PIX na maquininha para microempreendedores nos 9 primeiros meses de conta

2ª iniciativa: Campanha de ativação de Pix na maquininha para novos entrantes