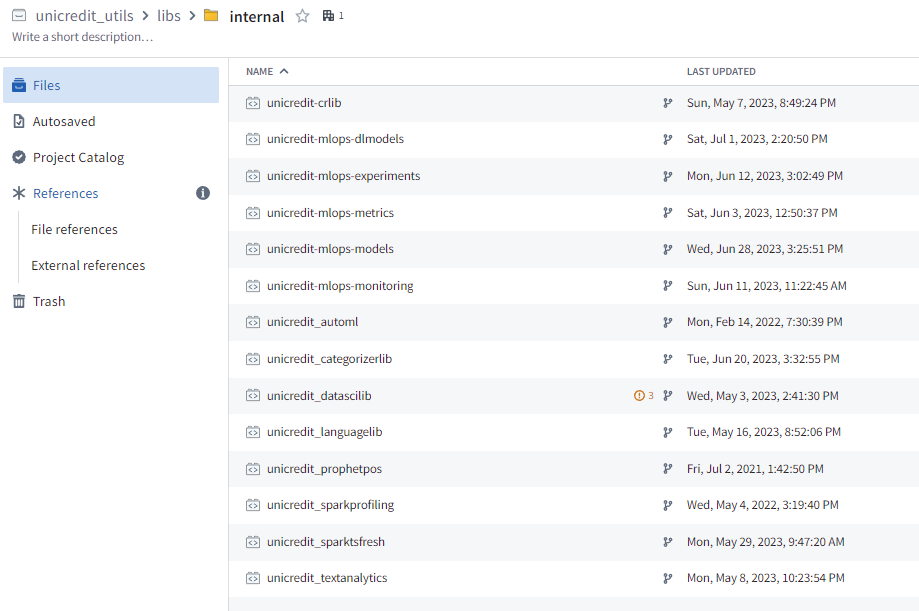
**Modello Cards**

Vedremo

* Valutazione Modello

1. Performance Modello
2. Analisi Variabili pù importanti

Mi sono scaricato alcune librerie che si sono creati loro:



**Valutazione Modello**

Le loro analisi Relative al modello di cards si trovano in:

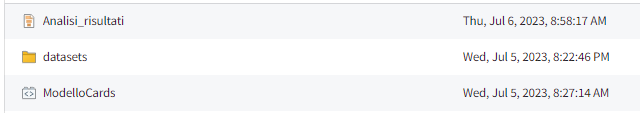
[**https://foundry.intranet.unicredit.eu/workspace/contour-app/ri.contour.main.analysis.2857e9b0-6bfc-4f60-91db-9b1467072bbd/path/ri.contour.main.ref.5d4c5aac-d27a-4652-a484-df06409075e2/board/db36813d-8aa1-3771-9ce1-43ea1edba87a?viewMode=edit**](https://foundry.intranet.unicredit.eu/workspace/contour-app/ri.contour.main.analysis.2857e9b0-6bfc-4f60-91db-9b1467072bbd/path/ri.contour.main.ref.5d4c5aac-d27a-4652-a484-df06409075e2/board/db36813d-8aa1-3771-9ce1-43ea1edba87a?viewMode=edit)

Mi sono salvato il dataset in cui Adriana ha applicato i modello nel repository in UR00601:

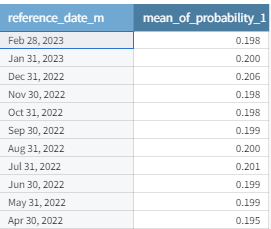




E mi sono fatto un Contour **Analisi\_risultati** dove analizzo questa tabella:



1. **Performance Modello**
2. La prima cosa loro hanno tanti snapshot ,**verificano la probabilità media se è sempre uguale** all’incidenza della variabile target , e se non ci sono distorsioni:



Come vedi la media della probabiità è uguale in tutti gli snpashot ed è circa il 19% che è l’incidenza della variabile target che è il 17%.

1. La seconda cosa da vedere è se **Auc è uguale per tutti snapashot**

from pyspark.sql import functions as F

from transforms.api import transform\_df, Input, Output

from pyspark.sql import types as T

from pyspark.sql.types import \*

import pandas as pd

@transform\_df(

    Output("/Users/UR00601/Modelli\_Gruppo\_Barardi/cards/datasets/Z01\_Application/A001A02\_Performance\_01"),

    df=Input("/Users/UR00601/Modelli\_Gruppo\_Barardi/cards/datasets/Z01\_Application/A001A01\_Modello\_Applicato"),

)

def compute(df):

    date\_col = 'reference\_date\_m'

    target\_col = 'target'

    prediction\_col = 'probability\_1'

    df = (

        df

        .select(

            'reference\_date\_m',

            'target',

            'probability\_1',

            'test\_type',

            'consent'

        )

    )

    udf\_out\_schema = T.StructType(

        [

            T.StructField(date\_col, T.DateType(), False),

            T.StructField('auc', T.FloatType(), True),

        ]

    )

    # pandas\_udf

    def pd\_udf(group):

        import pandas as pd  # noqa

        from sklearn.metrics import roc\_auc\_score

        if group[target\_col].sum() > 1:

            auc = roc\_auc\_score(group[target\_col], group[prediction\_col])

        else:

            auc = None

        results\_df = pd.DataFrame(

            [

                {

                    date\_col: group[date\_col].values[0],

                    'auc': auc

                }

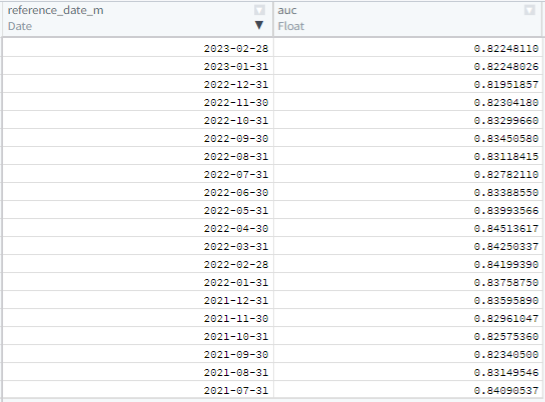
            ]

        )

        return results\_df

    results\_df = df.groupby(date\_col).applyInPandas(pd\_udf, udf\_out\_schema)

    return results\_df



**Come vedi è costante nel tempo.**

**Tutte le altre le metriche con calcolo Uplflit**

Vengono calcolati tutti gli ndicatori in questa query:

def unnamed(inference\_lgbm\_hyperopt\_featImp\_quantilePerf):

    n\_col = "n"

    ref\_dt = 'reference\_date\_m'

    target\_col = 'target'

    true\_positives\_col = "true\_positives"

    n\_cut\_col = "n\_cut"

    cum\_true\_positives\_col = "cum\_true\_positives"

    percent\_rank\_col = "percent\_rank"

    temp\_percentile\_col = "percentile\_"

    percentile\_col = 'percentile'

    quantile\_granularity = 100

    recall\_col = 'recall'

    precision\_col = 'precision'

    f1\_col = 'f1\_score'

    uplift\_col = 'uplift'

    baseline\_incidence = 'baseline\_incidence'

    overall\_samples = 'overall\_ndgs'

    overall\_pos = 'overall\_pos'

    df = (

        inference\_df

        .select(

            'ndg',

            ref\_dt,

            target\_col,

            prediction\_col,

            F.percent\_rank().over(Window.partitionBy('reference\_date\_m').orderBy(prediction\_col)).alias(percent\_rank\_col),

        )

        .withColumn(temp\_percentile\_col,(F.col(percent\_rank\_col) \* quantile\_granularity).cast(IntegerType()))

        .withColumn(percentile\_col,

            F.when(

                F.col(temp\_percentile\_col) == quantile\_granularity,

                F.lit(quantile\_granularity - 1),

            ).otherwise(F.col(temp\_percentile\_col)),

        )

        .drop(temp\_percentile\_col)

    )

    percentile\_counts = (

        df

        .groupby(ref\_dt, percentile\_col)

        .agg(

            F.count("\*").alias(n\_col),

            F.sum(F.col(target\_col)).alias(true\_positives\_col)

        )

    )

    w = Window.partitionBy(ref\_dt).orderBy(F.desc(percentile\_col))

    w\_date = Window.partitionBy(ref\_dt)

    metrics\_df = (

        percentile\_counts

        .withColumn(n\_cut\_col, F.sum(F.col(n\_col)).over(w))

        .withColumn(

            cum\_true\_positives\_col, F.sum(F.col(true\_positives\_col)).over(w)

        )

        .withColumn(

            recall\_col, F.col(cum\_true\_positives\_col) / F.sum(F.col(true\_positives\_col)).over(Window.partitionBy(ref\_dt))

        )

        .withColumn(

            precision\_col, F.col(cum\_true\_positives\_col) / F.col(n\_cut\_col)

        )

        .withColumn(

            f1\_col,

            2.0

            \* (F.col(precision\_col) \* F.col(recall\_col))

            / (F.col(precision\_col) + F.col(recall\_col)),

        )

        .withColumn(overall\_samples, F.sum(F.col(n\_col)).over(w\_date))

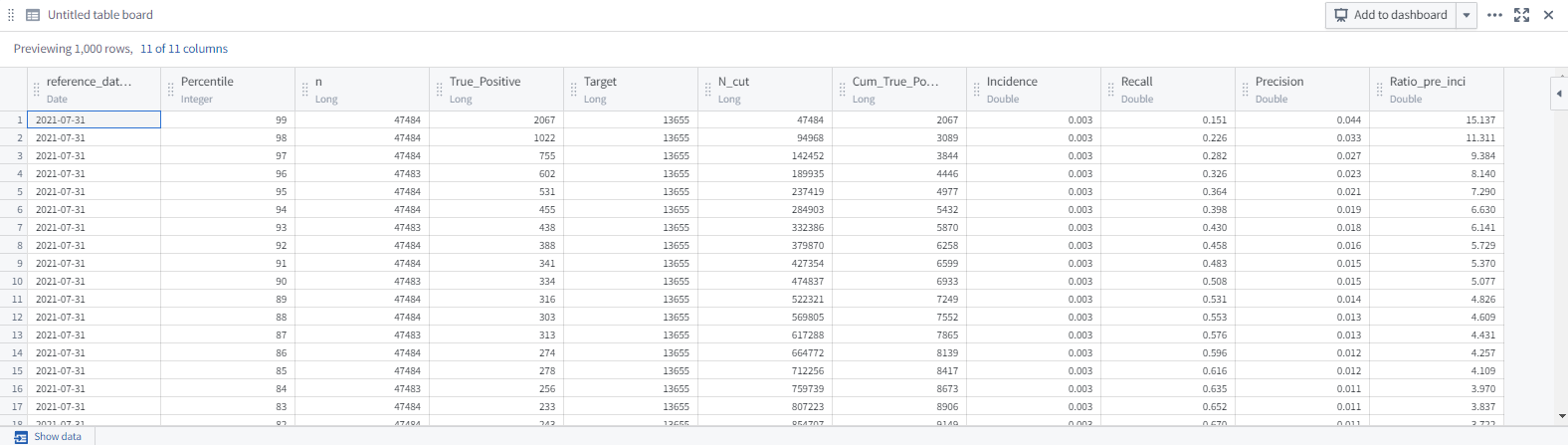
        .withColumn(overall\_pos, F.sum(F.col(true\_positives\_col)).over(w\_date))

        .withColumn(baseline\_incidence, F.col(overall\_pos)/F.col(overall\_samples))

        .withColumn(uplift\_col, F.col(precision\_col)/F.col(baseline\_incidence))

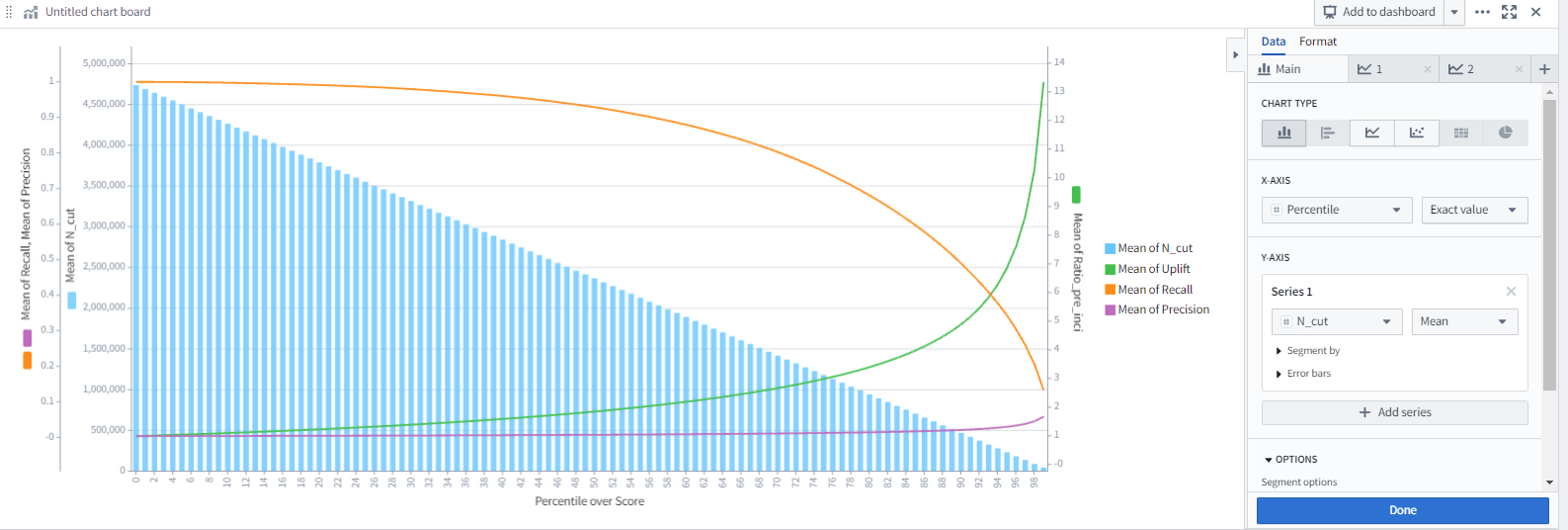
        .orderBy(F.col(percentile\_col).desc())

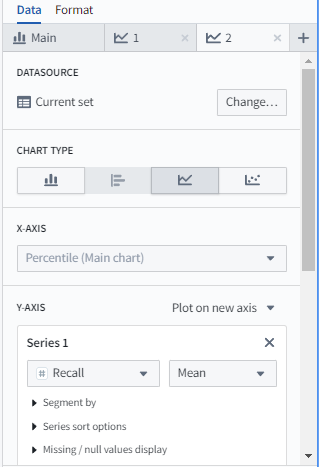
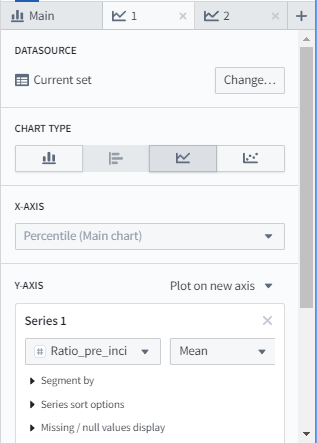
    )

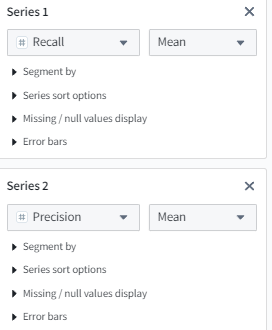


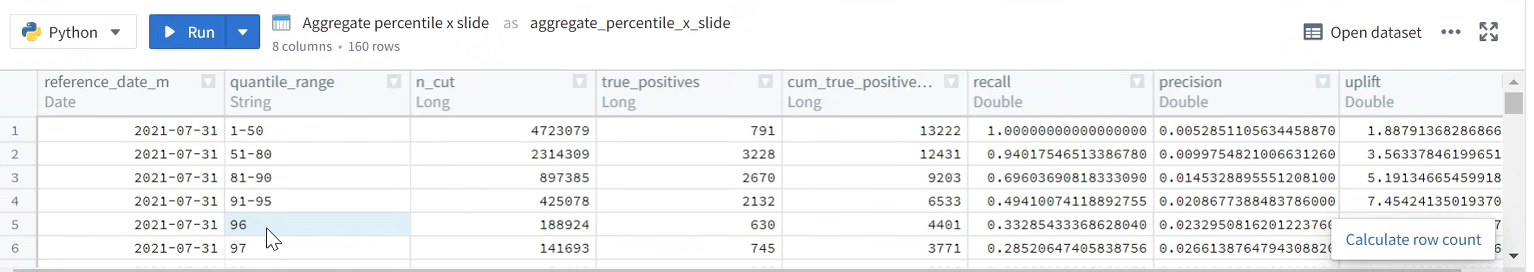
**Qui è excel scaricato** ****

**E da qui parte upfilt**









**Funzioni che calcolano il dataset :**

# Functions defined here will be available to call in

# the code for any table.

from pyspark.sql import Window, functions as F

from pyspark.sql import types as T

from pyspark.sql.types import \*

import matplotlib.pyplot as plt

import numpy as np

import lightgbm as lgb

from sklearn.model\_selection import train\_test\_split

from pyspark.ml.feature import QuantileDiscretizer

import datetime

import itertools

from sklearn.metrics import auc, precision\_recall\_curve, recall\_score, precision\_score, roc\_auc\_score

import pickle

import pandas as pd

Q = 90

METRICS = [

    'recall',

    'precision',

    'baseline\_uplift',

    'target\_uplift'

    ]

schema = StructType(

        [

            StructField('reference\_date\_m', DateType(), False),

            StructField('partition', StringType(), False),

            StructField(f'{Q}\_quantile\_recall', FloatType(), True),

            StructField(f'{Q}\_quantile\_precision', FloatType(), True),

            StructField('auc', FloatType(), True),

            StructField('detected\_positives', IntegerType(), False),

            StructField('tot\_positives', IntegerType(), False),

            StructField('mean\_target', FloatType(), False),

            StructField('mean\_pred', FloatType(), False),

            StructField('count', IntegerType(), False),

            StructField(f'{Q}\_count', IntegerType(), True)

        ]

    )

# PandasUDF that trains the lgb Classifier on a specific fold, on the train partition and the applies on the val set

def evaluate\_fold(group):

    from sklearn.metrics import roc\_auc\_score

    import pandas as pd  # noqa

    group = group.sort\_values('pred')

    # create a (unique) ranking columns that follows the sorting computed above

    group['rank'] = range(group.shape[0])

    # Compute quantiles over the ranking column, in order to group the predictions

    group['quantile'] = pd.qcut(group['rank'], 1000, labels=False)

    if group['target'].sum() > 1:

        auc = roc\_auc\_score(group['target'], group['pred'])

    else:

        auc = None

    mean\_target = group['target'].mean()

    mean\_pred = group['pred'].mean()

    # Compute metrics with such support:

    # Detected Positive: all the samples included in the top quantile(s)

    # True Positive: all the samples with target = 1 in the top quantile(s)

    # Tot Positive: all the samples with target = 1 in the validation set

    tot\_pos = group['target'].sum()

    best\_quantile = group[group['quantile'] >= Q]['target']

    detected\_pos = best\_quantile.sum()

    # Recall = Detected Positive / Tot Positive

    recall = detected\_pos / tot\_pos

    # Precision = True Positive / Detected Positive

    precision = best\_quantile.mean()

    results\_df = pd.DataFrame(

        [

            {

                'reference\_date\_m': group['reference\_date\_m'].values[0],

                'partition': group['partition'].values[0],

                f'{Q}\_quantile\_recall': recall,

                f'{Q}\_quantile\_precision': precision,

                'auc': auc,

                'detected\_positives': detected\_pos,

                'tot\_positives': tot\_pos,

                'mean\_target': mean\_target,

                'mean\_pred': mean\_pred,

                'count': group.shape[0],

                f'{Q}\_count': len(best\_quantile)

            }

        ]

    )

    return results\_df

def AUPRC(y\_true, y\_pred, pos\_label=1):

    precision, recall, \_ = precision\_recall\_curve(y\_true, y\_pred, pos\_label=pos\_label)

    return auc(recall, precision)

def AUPRC\_lgbm(y\_true, y\_pred):

    '''

    https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html

    Custom eval function expects a callable with following signatures:

        - func(y\_true, y\_pred)

        - func(y\_true, y\_pred, weight)

        - func(y\_true, y\_pred, weight, group)

    returns:

        - (eval\_name: str, eval\_result: float, is\_higher\_better: bool)

        - list of (eval\_name: str, eval\_result: float, is\_higher\_better: bool)

    '''

    return 'AUPRC', AUPRC(y\_true, y\_pred), True

def metrics\_backbone\_calculation(inference\_df, prediction\_col):

        n\_col = "n"

        ref\_dt = 'reference\_date\_m'

        target\_col = 'target'

        true\_positives\_col = "true\_positives"

        n\_cut\_col = "n\_cut"

        cum\_true\_positives\_col = "cum\_true\_positives"

        percent\_rank\_col = "percent\_rank"

        temp\_percentile\_col = "percentile\_"

        percentile\_col = 'percentile'

        quantile\_granularity = 100

        recall\_col = 'recall'

        precision\_col = 'precision'

        f1\_col = 'f1\_score'

        uplift\_col = 'uplift'

        baseline\_incidence = 'baseline\_incidence'

        overall\_samples = 'overall\_ndgs'

        overall\_pos = 'overall\_pos'

        df = (

            inference\_df

            .select(

                'ndg',

                ref\_dt,

                target\_col,

                prediction\_col,

                F.percent\_rank().over(Window.partitionBy('reference\_date\_m').orderBy(prediction\_col)).alias(percent\_rank\_col),

            )

            .withColumn(temp\_percentile\_col,(F.col(percent\_rank\_col) \* quantile\_granularity).cast(IntegerType()))

            .withColumn(percentile\_col,

                F.when(

                    F.col(temp\_percentile\_col) == quantile\_granularity,

                    F.lit(quantile\_granularity - 1),

                ).otherwise(F.col(temp\_percentile\_col)),

            )

            .drop(temp\_percentile\_col)

        )

        percentile\_counts = (

            df

            .groupby(ref\_dt, percentile\_col)

            .agg(

                F.count("\*").alias(n\_col),

                F.sum(F.col(target\_col)).alias(true\_positives\_col)

            )

        )

        w = Window.partitionBy(ref\_dt).orderBy(F.desc(percentile\_col))

        w\_date = Window.partitionBy(ref\_dt)

        metrics\_df = (

            percentile\_counts

            .withColumn(n\_cut\_col, F.sum(F.col(n\_col)).over(w))

            .withColumn(

                cum\_true\_positives\_col, F.sum(F.col(true\_positives\_col)).over(w)

            )

            .withColumn(

                recall\_col, F.col(cum\_true\_positives\_col) / F.sum(F.col(true\_positives\_col)).over(Window.partitionBy(ref\_dt))

            )

            .withColumn(

                precision\_col, F.col(cum\_true\_positives\_col) / F.col(n\_cut\_col)

            )

            .withColumn(

                f1\_col,

                2.0

                \* (F.col(precision\_col) \* F.col(recall\_col))

                / (F.col(precision\_col) + F.col(recall\_col)),

            )

            .withColumn(overall\_samples, F.sum(F.col(n\_col)).over(w\_date))

            .withColumn(overall\_pos, F.sum(F.col(true\_positives\_col)).over(w\_date))

            .withColumn(baseline\_incidence, F.col(overall\_pos)/F.col(overall\_samples))

            .withColumn(uplift\_col, F.col(precision\_col)/F.col(baseline\_incidence))

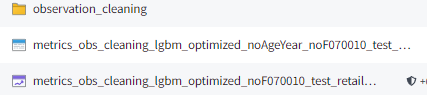
            .orderBy(F.col(percentile\_col).desc())

        )

        return metrics\_df

Questo è il dataset

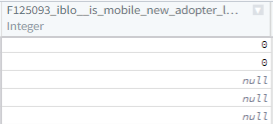




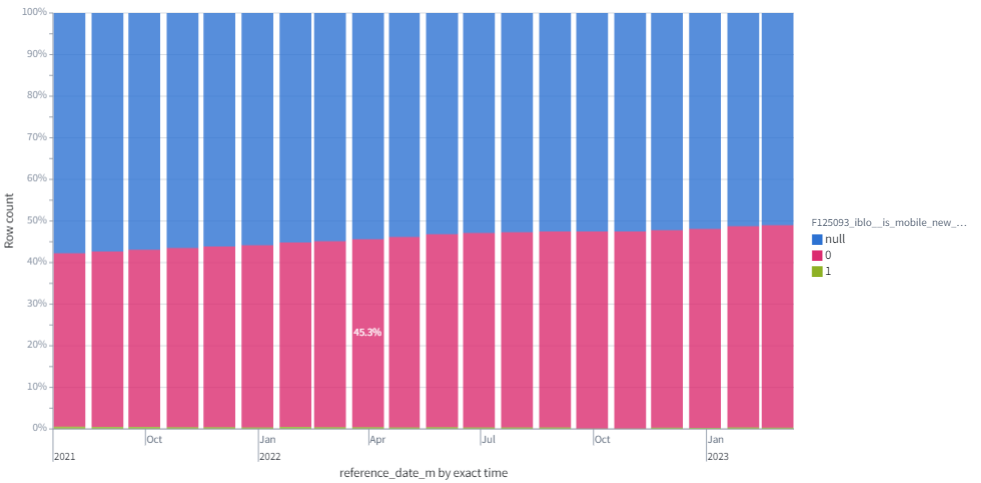


1. **Analisi delle variabili più importanti:**

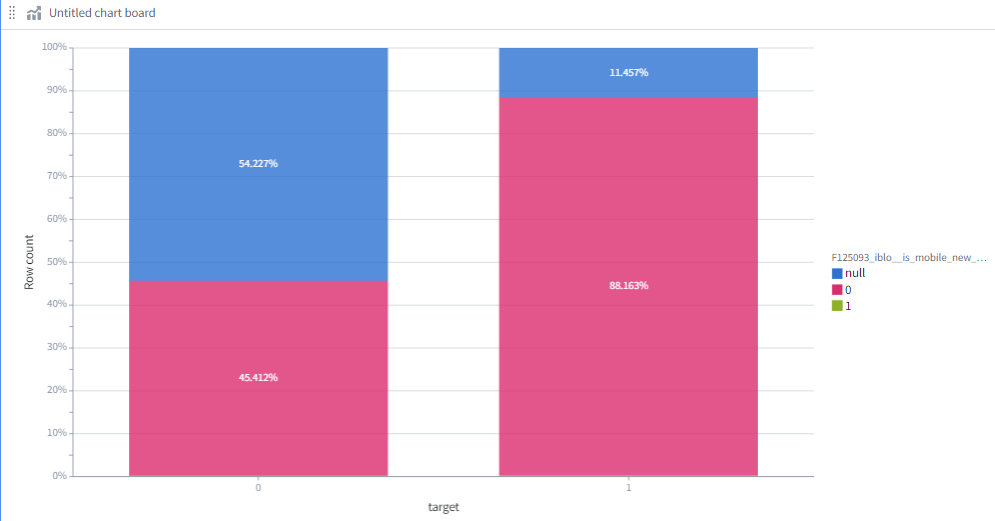
La variabile più importante del modello è:



La prima cosa da vedere è la distribuzione nei vari mesi:



La seconda cosa da vedere è la distribuzione di questa variabile con la variabile **target**



Importante se ti calcoli anche gli **shap values** ,puoi farti i grafici anche tipo Laura:

esempio ho la variabile età , la bachettizzo:

**CASE** **WHEN** F008532\_cntp\_\_age\_years >=18 **AND** F008532\_cntp\_\_age\_years <26 **THEN** '18-25'

**WHEN** F008532\_cntp\_\_age\_years >=26 **AND** F008532\_cntp\_\_age\_years <35 **THEN** '26-35'

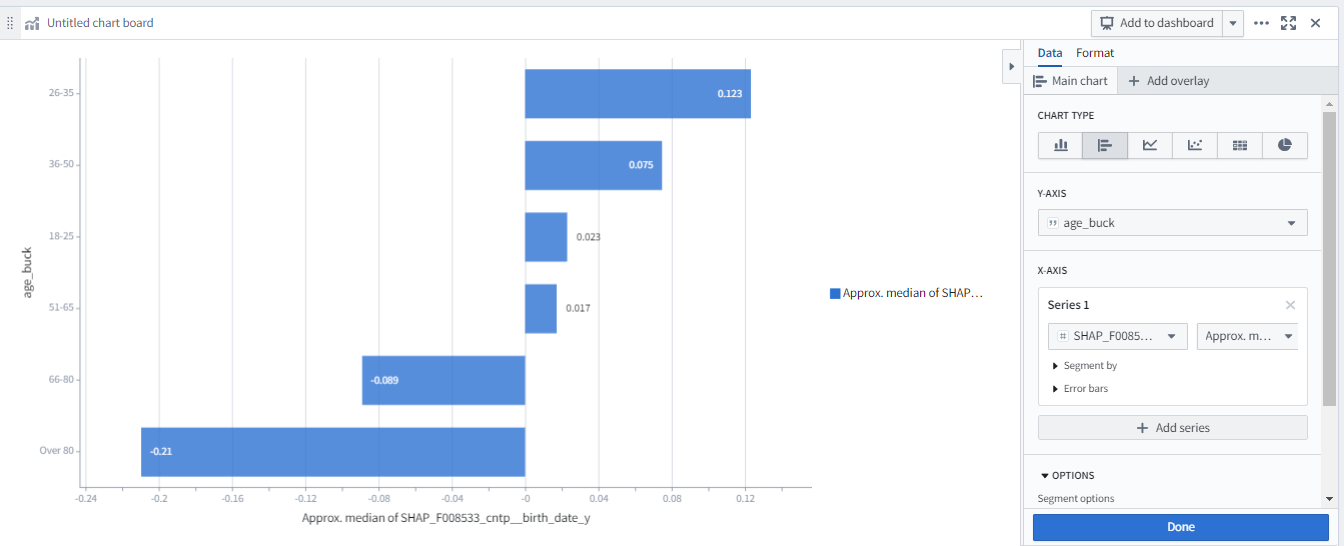
**WHEN** F008532\_cntp\_\_age\_years >=36 **AND** F008532\_cntp\_\_age\_years <51 **THEN** '36-50'

**WHEN** F008532\_cntp\_\_age\_years >=51 **AND** F008532\_cntp\_\_age\_years <66 **THEN** '51-65'

**WHEN** F008532\_cntp\_\_age\_years >=66 **AND** F008532\_cntp\_\_age\_years <81 **THEN** '66-80'

**ELSE** 'Over 80' **END**

Ho la relativa variabile con i valori shap che è SHAP\_F008533\_cntp\_\_birth\_date\_y



E fai l’approx median