**En la carpeta DISCOVERY**

There are two folder,

**- examples\_oficial:** that contains oficial notebooks of gurobi machine learning

**- examples\_mine\_discovery:** that contains my codes writes to understand how works gurobi-ml. Developed in base of oficial notebooks

**LINKS:**

- user guide gurobi ml: https://gurobi-machinelearning.readthedocs.io/en/stable/userguide.html

- link examples gurobi ml: https://gurobi-machinelearning.readthedocs.io/en/stable/auto\_examples/index.html

- link example optimization + clasification model: https://gurobi-machinelearning.readthedocs.io/en/stable/auto\_examples/example2\_student\_admission.html#sphx-glr-auto-examples-example2-student-admission-py

- link example simple optimization + regressors model: https://gurobi-machinelearning.readthedocs.io/en/stable/auto\_examples/example4\_price\_optimization.html#sphx-glr-auto-examples-example4-price-optimization-py

- GUROBI REFERENCE GUIDE 1000 PAGES: <https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/10.0/refman.pdf>

- GUROBI GITHUB FULL EXAMPLES: <https://github.com/Gurobi/modeling-examples>

**ZOOM OPTIMIZATION + REGRESSORS MODELS:**

- Github repo: <https://github.com/Gurobi/modeling-examples/tree/master/price_optimization>

- Notebooks colab (the notebook in github are the same):

- Create a restricction that show the relation price/demand: <https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/price_optimization/price_optimization.ipynb#scrollTo=3f7ff154>

- Create a restricction price/demand using sklearn - gurobi ml (free licence): <https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/price_optimization/price_optimization_gurobiML.ipynb>

- Create a restricction price/demand using sklearn - gurobi ml (pay licence): <https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/price_optimization/price_optimization_gurobiML_wls.ipynb#scrollTo=pzQR12rAr-b_>

**ORDEN DEL REPO**

0\_oficial\_examples: folder que contiene los ejemplos de gurobi para entender cómo utilizar los diferentes packages de gurobi. Luego, el resto de folder están destinados

1\_data: generar data para ser utilizadas en los ejemplos. Se sigue el código de referencia de gurobi para generar el dataset

2\_eda: análisis exploratorio de datos básico para entender un poco el dataset

3\_feature\_eng: Transformaciones de features para entrenar modelo. Transformaciones desde la data raw obtenida en folder 1\_data

4\_model\_ml: códigos para entrenar modelos de ml y evaluarlos. Se entrenan 2 versiones, un solo modelo y varios modelos individuales

5\_optimization: generar modelo de optimización de gurobi a partir de los modelos predictores de machine learning

Artifacts: folder where differents are saved. Such as, data, models, etc

**COMBINAR MODELOS DE MACHINE LEARNING CON OPTIMIZACION – GUROBI – AGREGAR LA PREDICCIÓN DE UN MODELO DE MACHINE LEARNING COMO RESTRICCIONES DEL MODELO DE OPTIMIZACION**

Para agregar un modelo de ML como restricción en gurobi se utiliza el siguiente método de gurobi

add\_predictor\_constr(m, ml\_model, m\_feats, d)

donde:

m: modelo de optimización de gurobi

ml\_model: modelo de machine learning – recibe un input y genera un output

m\_feats: input que recibe el modelo de machine learning. Cada fila representa una instancia del modelo de ML y la cantidad de filas debe ser consistente con la cantidad del filas de la variable de decisión “d”. Además “m\_feats” contiene “features que son variables de decisión” del modelo de optimización

d: predicción del modelo de machine learning. Es una variable de decisión del modelo de optimización. En el ejemplo es la demanda/ventas dado un modelo de machine learning que ve el precio y otras features no variables de decisión. La variable de decisión ventas es por región d(r) pro lo tanto se predicen N regiones y m\_feats debe tener la cantidad de filas equivalente a las r regiones

ADEMÁS, como se ve, no se aceptan un pipeline de features engineering aparte, se recibe datos y un modelo que predice con esos datos. Así, para hacer feature engineering se debe:

- Hacer el feature engineering del modelo antes y pasarle al optimizador la instancia de predicción ya procesada

- Utilizar el objeto Pipeline de sklearn y generar un artefacto que tiene el feature engineering y el modelo

La elección va a depender principalmente, en la cantidad de datos a procesar en el feature engineering y la mejor forma de tener el menor tiempo de procesamiento. Por ejemplo hacer el feature engineering en bigquery y tener un procesamiento de datos más rápido, luego generar la instancia input del modelo de optimización

- SE ENTRENARON 6 TIPOS DISTINTOS DE MODELOS Y SE EVALUARON POR REGIÓN

- especificar tipo de modelo y tipo de dataset

- mostrar tablas con los resultados

- mostrar tablas con los gráficos

Por la licencia de gurobi, aunque los modelos de gradient boosting den mejores métricas. La licencia gratuita no permite incluir dichos modelos y para probar gurobi se eligieron los mejores resultados de regresiones lineales

Variación en los modelos

En un primer grupo se divide de acuerdo a los datos utilizados en el entrenamiento, específicamente las features utilizadas. Manteniendo el target de predecir la demanda en la región X. Los datos son los siguientes:

- basic\_features: consiste en el dataset de ejemplo con las features del ejemplo original de gurobi, con una pequeña modifación donde se elimina el año.

Features: región, peak, Price

- Price\_regions: consiste en una transformación del dataset de ejemplo donde en el original cada región tenías un precio y cada uno era una fila y ahora para predecir la demanda de la región X se utiliza el precio de cada una de las regiones. Esto bajo la idea económica de vienes sustitutos y complementarios (aunque es un solo producto segmentado el precio en diferentes mercados/regiones en este caso)

Features: [‘region’, 'peak', 'price\_great\_lakes', 'price\_midsouth', 'price\_northeast', 'price\_northern\_new\_england', 'price\_plains', 'price\_southcentral', 'price\_southeast', 'price\_west']

Por otro lado, se entrenan diferentes modelos variando el tipo de modelo y la cantidad de modelos. Se probaron 3 variaciones

- one linear regression: Ajustar una regresión lineal para predecir el precio utilizando el dataset de entrada

- multiple linear regression: Ajustar multiples regresiones lineales, una por región. Si se observa en ambos datasets probados, existe una feature categórica región para indicar qué región es. En este caso se entrena un modelo de regresión para cada región

- gradient boosting regresor: Se entrena un modelo más complejo, siguiendo la idea original de gurobi. El problema de esto es que para poder tener un modelo de optimización con gurobi y utilizando gbr se necesita licencia de gurobi por el tamaño del modelo

Así se entrenaron 6 diferentes “modelos” para predecir el precio

|  |  |  |
| --- | --- | --- |
| Notebook de entrenamiento | Dataset | Modelo |
| 1\_basic\_features\_one\_lr | basic\_features | one linear regression |
| 2\_basic\_features\_multiple\_lr | basic\_features | multiple linear regression |
| 3\_basic\_features\_one\_gbr | basic\_features | gradient boosting regresor |
| 4\_prices\_regions\_one\_lr | Price\_regions | one linear regression |
| 5\_prices\_regions\_multiple\_lr | Price\_regions | multiple linear regression |
| 6\_prices\_regions\_one\_grb | Price\_regions | gradient boosting regresor |

**COMPARACIÓN RESULTADOS METRICAS GLOBALES DATASET**

**1\_basic\_features\_one\_lr**

R2

r2\_train: 0.878

r2\_test: 0.881

MAE

mae\_train: 0.546

mae\_test: 0.542

MSE

mse\_train: 0.526

mse\_test: 0.496

RMSE

rmse\_train: 0.725

rmse\_test: 0.704

**2\_basic\_features\_multiple\_lr**

R2

r2\_train: 0.864

r2\_test: 0.872

MAE

mae\_train: 0.563

mae\_test: 0.532

MSE

mse\_train: 0.587

mse\_test: 0.532

RMSE

rmse\_train: 0.766

rmse\_test: 0.729

**3\_basic\_features\_one\_gbr**

R2

r2\_train: 0.898

r2\_test: 0.892

MAE

mae\_train: 0.47

mae\_test: 0.483

MSE

mse\_train: 0.442

mse\_test: 0.449

RMSE

rmse\_train: 0.665

rmse\_test: 0.67

**4\_prices\_regions\_one\_lr**

R2

r2\_train: 0.883

r2\_test: 0.889

MAE

mae\_train: 0.529

mae\_test: 0.517

MSE

mse\_train: 0.506

mse\_test: 0.46

RMSE

rmse\_train: 0.711

rmse\_test: 0.678

**5\_prices\_regions\_multiple\_lr**

R2

r2\_train: 0.917

r2\_test: 0.911

MAE

mae\_train: 0.427

mae\_test: 0.438

MSE

mse\_train: 0.361

mse\_test: 0.37

RMSE

rmse\_train: 0.601

rmse\_test: 0.608

**6\_prices\_regions\_one\_grb**

R2

r2\_train: 0.92

r2\_test: 0.916

MAE

mae\_train: 0.401

mae\_test: 0.423

MSE

mse\_train: 0.344

mse\_test: 0.348

RMSE

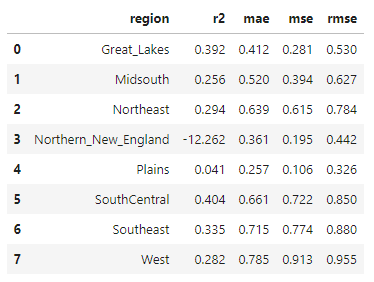
rmse\_train: 0.587

rmse\_test: 0.59

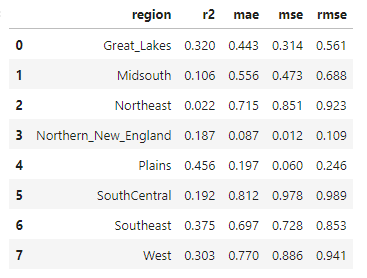
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1\_basic\_features\_one\_lr** | **2\_basic\_features\_multiple\_lr** | **3\_basic\_features\_one\_gbr** | **4\_prices\_regions\_one\_lr** | **5\_prices\_regions\_multiple\_lr** | **6\_prices\_regions\_one\_grb** |
| r2\_train | 0.878 | 0.864 | 0.898 | 0.883 | 0.917 | 0.92 |
| r2\_test | 0.881 | 0.872 | 0.892 | 0.889 | 0.911 | 0.916 |
| mae\_train | 0.546 | 0.563 | 0.47 | 0.529 | 0.427 | 0.401 |
| mae\_test | 0.542 | 0.532 | 0.483 | 0.517 | 0.438 | 0.423 |
| mse\_train | 0.526 | 0.587 | 0.442 | 0.506 | 0.361 | 0.344 |
| mse\_test | 0.496 | 0.532 | 0.449 | 0.46 | 0.37 | 0.348 |
| rmse\_train | 0.725 | 0.766 | 0.665 | 0.711 | 0.601 | 0.587 |
| rmse\_test | 0.704 | 0.729 | 0.67 | 0.678 | 0.608 | 0.59 |

**COMPARACIÓN RESULTADOS METRICAS PARA CADA DATASET DE CADA REGION – conjunto de test**

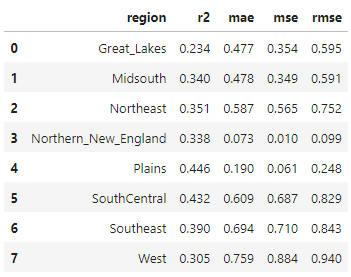
**1\_basic\_features\_one\_lr**



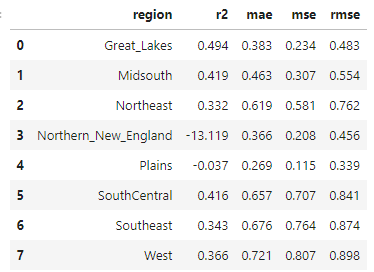
**2\_basic\_features\_multiple\_lr**



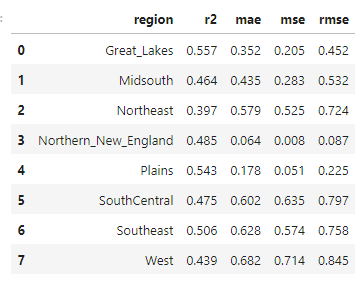
**3\_basic\_features\_one\_gbr**



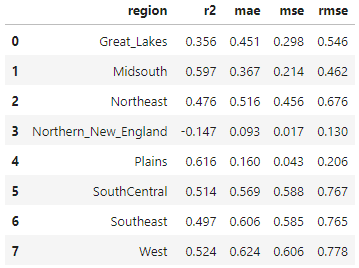
**4\_prices\_regions\_one\_lr**



**5\_prices\_regions\_multiple\_lr**



**6\_prices\_regions\_one\_grb**



**RESUMEN POR REGIÓN Y MÉTRICA RMSE TEST DE CADA MODELO**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1\_basic\_features\_one\_lr** | **2\_basic\_features\_multiple\_lr** | **3\_basic\_features\_one\_gbr** | **4\_prices\_regions\_one\_lr** | **5\_prices\_regions\_multiple\_lr** | **6\_prices\_regions\_one\_grb** |
| Great\_Lakes | 0.530 | 0.561 | 0.595 | 0.483 | 0.452 | 0.546 |
| Midsouth | 0.627 | 0.688 | 0.591 | 0.554 | 0.532 | 0.462 |
| Northeast | 0.784 | 0.923 | 0.752 | 0.762 | 0.724 | 0.676 |
| Northern\_New\_England | 0.442 | 0.109 | 0.099 | 0.456 | 0.087 | 0.130 |
| Plains | 0.326 | 0.246 | 0.248 | 0.339 | 0.225 | 0.206 |
| SouthCentral | 0.850 | 0.989 | 0.829 | 0.841 | 0.797 | 0.767 |
| Southeast | 0.880 | 0.853 | 0.843 | 0.874 | 0.758 | 0.765 |
| West | 0.955 | 0.941 | 0.940 | 0.898 | 0.845 | 0.778 |

MODELO DE OPTIMIZACIÓN DE GUROBI- Optimize for Price and Supply of Avocados

Para el modelo de optimización de gurobi, se van a probar dos de los modelos desarrollados en el ejemplo, uno donde se utiliza solo un modelo y otro donde se emplean múltiples modelos en gurobi

En en el ejemplo se cargan solo regresiones lineales pero gurobi acepta multiples tipos de modelos de distintos packages

Link con todos los modelos soportados: <https://gurobi-machinelearning.readthedocs.io/en/stable/api.html>

- In addition, to define the decision variables, parameters, restriction, etc of the optimization model are used "gurobipy-pandas". Using this package is possible define the optimization model using pandas DataFrames

Importante las features que van a cambiar de valor y el target del modelo de machine learning, ambos son variables de decisión para el modelo de optimización

Importante, al definir variables de decisión, por defecto los límies son 0 e infinito. Si se desea que la variable tome valores distinto, se deben definir.

**MUY IMPORTANTE – FEATURE ENG Y MODELOS**

Los pasos clásicos para tener un modelo entrenado es primero hacer el feature eng que corresponda y luego entrenar el modelo.

Hasta ahora ya se tiene claridad de la forma en que se integra un modelo de machine learning en un modelo de optimización de gurobi, pero el procesamiento de los datos, feauture eng, que a partir de los datos “reales” genera los datos “de entrada” que recibe el modelo de machine learning, está la duda de donde van. Feature eng tales como escalar los datos, aplicar logaritmos, exponencial, etc

Ahora bien, pensar por ejemplo que se tiene un modelo que dado el precio predice la demanda. Tanto precio como demanda son variables de decisión del modelo de optimización y el modelo de machine learning tiene que por ejemplo hacer transformación logarítmica del precio, luego escalar y además hacer otras transformaciones a otras features que no son variables de decisión del optimizador.

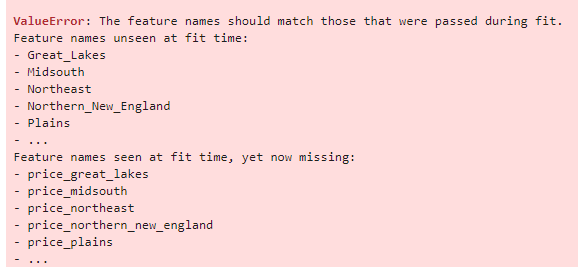
El precio (feature del modelo de machine learning) como ya se dijo es una variable de decisión y se utiliza en otras restricciones del modelo de optimización, entonces, la escala de la variable de decisión precio tiene ser la misma y si el modelo de ML recibe en realidad un precio transformado, **esta transformación tiene que estar empaquetada en el artefacto (feature eng + modelo).**

**Otra solución, no probada, podría ser crear una segunda variable de decisión precio\_transform**, así se tiene una restricción que recibe el precio, llama el artefacto que hace la transformación de los datos (precio -> artifact feature eng -> precio\_transform) y luego de tener un precio transformado llamar al modelo (precio\_transform -> model -> demanda) y con eso se podría mantener el feature eng separado del modelo. Lo más lógico sería empaquetar en un solo artefacto, pero **si por ejemplo el feature eng se hace en sklearn y se entrena una red neuronal en keras, se vuelve obligatorio tener 2 artefactos separados y esta aplicación permitiría solucionar el problema**

**MUY IMPORTANTE DEFINIR NOMBRE DE LAS FEATURES IGUALES EN ENTRENAMIENTO Y EN INFERENCIA**

Los nombres de las features del dataframe de entrenamiento del modelo de machine learning y el nombre de las features en el dataframe con la instancia de inferencia para el optimizador deben de tener el MISMO NOMBRE DE LAS COLUMNAS (además del mismo orden que es lo obvio) porque sino gurobi machine learning retorna un error

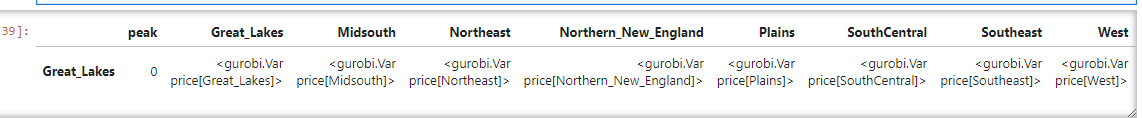
Por ejemplo se puede retornar el siguiente error



Esto porque el dataframe con los datos de entrenamiento tenía la siguiente estructura (nombre de las columnas):



Mientras que el dataframe con la inferencia para el optimizador tiene la siguiente estructura (nombre de las columnas):



Se observa que el nombre de las columnas no es coincidente

**EJEMPLOS GUROBI ML DESARROLLADOS**

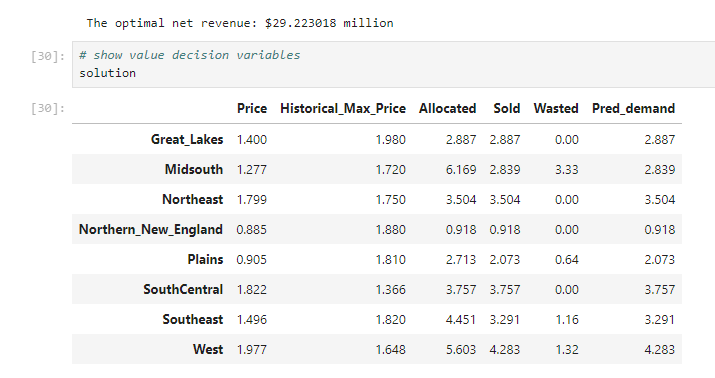
1. Un modelo que dado una feature que es variable de decisión predecir la demanda que es otra variable de decisión y optimizar.

Es relativamente fácil probar mover manualmente modificar la única variable de decisión y encontrar el “punto óptimo”. Sin embargo, esto se complica cuando existen múltiples modelos y cada uno ve múltiples variables de decisión porque el probar a “mano” las diferentes combinaciones para obtener la solución óptima es muy difícil por no decir imposible

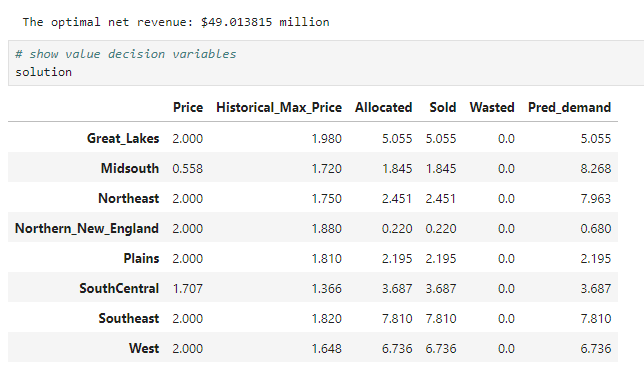
1. Múltiples modelos donde cada modelo predice la demanda de una región y recibe como input 8 variables de decisión, el precio de cada una de las regiones. Entonces existen múltiples combinaciones que no se pueden probar a mano
2. Mismo caso anterior, exactamente, incluso los mismos modelos pero modificado el problema de optimización para optimizar en un periodo de tiempo t. Esto se hace invocando el modelo de machine learning múltiples veces en cada uno de los tiempos t

En el caso de los ejemplos 1 y 2, son casos comparables, donde únicamente en lugar de tener un único modelo de machine learning para predecir la demanda de todas las regiones, se crearon múltiples modelos uno para predecir la demanda de cada región y que veían los precios de diferentes regiones. Luego de resolver el mismo problema de optimización, modificando únicamente los modelos de ml ya mencionados, estos fueron los resultados

Optimizador 1 – un modelo de ML



Optimizador 2 – múltiples modelos de ML



Se observa que el optimizador retorna una solución óptima por un valor económico el doble, solo con modelos que pueden predecir de mejor forma (obteniendo mejores métricas)

OPTIMIZADOR 3 – NO COMPARABLE – OPTIMIZAR MULTIPLES PERIDOOS DE TIEMPO

DOCUMENTATION ATRITBUTOS DE CADA UNA DE LAS CALSES DE GUROBI

<https://www.gurobi.com/documentation/current/refman/attributes.html#sec:Attributes>

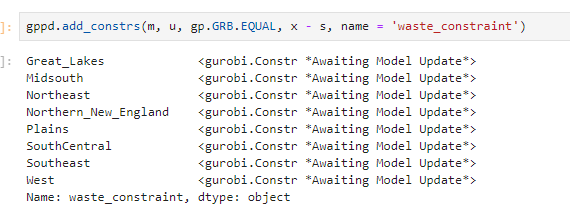
**IMPORTANTE DEFINIR RESTRICCIONES GUROBI PARA TODO ELEMENTO DEL SET.**

Al definir una restricción para todo elemento del set solo basta con definir la inecuación de la restricción de tal forma que el largo de restricciones del lado izquierdo y lado derecho sean de la misma dimensionalidad. Solo con eso gurobi entenderá que es una restricción para cada elemento del conjunto.

Por ejemplo,



gppd.add\_constrs(m, u, gp.GRB.EQUAL, x - s, name = 'waste\_constraint')



En el jupyter notebook al definir una restricción se muestra la cantidad de restricciones inidividuales que se crearon con ese código (En este caso, como se puede observar se crea una restricción para cada uno de los elementos del conjunto región R)

Además, después de definir todas las restricciones es necesario “compilar” llamando el método model.update()

**IMPORTANTE:**

También se puede conocer el lado izquierdo de la restricción

<https://support.gurobi.com/hc/en-us/articles/9424726080529-How-do-I-access-the-left-hand-side-of-a-constraint>



**IMPORTANTE PONERLE NOMBRE A LAS CONSTRAINTS Y A LAS VARIABLES DE GUROBI PARA DESPUÉS EN CASO DE QUERER DEBUGEAR LOS VALORES SE PUEDA HACER**

**Ahora, falta saber:**

- Obviamente, saber cómo estas predicciones en un entorno en vivo se comportaron y si realmente realizando las acciones tomandas por el optimizador. Eso no se puede saber a menos que se ponga en vivo

- Pero, de forma offline, con los datos, se puede ver si la solución propuesta por el optimizador, específicamente las variables de decisión que son features de un modelo de machine learning, poder saber si habían datos o no. Porque podría ser que la solución propuesta sea una generalización y que no habían datos cercanos (y que en parte está bien porque el modelo tiene que ser capaz que tener el comportamiento intrínseco del proceso que está modelando), pero si no hay datos cercanos de la solución del optimizador, no hay una contraparte histórica que respalde la solución del modelo

# CONOCER EL STATUS DEL MODELO DE GUROBI Y SI FALLÓ O NO

#### know the status of the model - 2 a optimal solution was founded

# docu: https://www.gurobi.com/documentation/current/refman/optimization\_status\_codes.html#sec:StatusCodes

model\_opt.Status