openGauss AI特性创新实践课



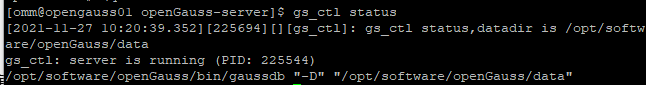
华为技术有限公司

# 关卡一、openGauss数据安装及基本操作

openGauss数据安装及基本操作, 作业提交任务如下：

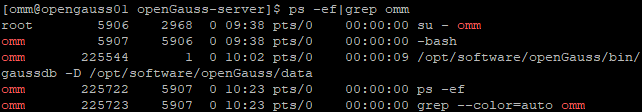
任务一：数据库状态验证

1. 查询数据库状态成功截图



任务二：数据库服务进程验证

1. 查看数据库服务进程截图（包含数据库服务器的主机名）



实验思考题：为什么需要通过源码编译，安装数据库？

答：源码编译安装的方式，原因有：

1、满足不同的运行平台，Linux版本很多，每个版本所要求的软件和内核都不一样，而源码安装更容易满足各个平台的依赖环境，而且在编译过程，部分程序是可以根据当前环境配置进行优化。

2、方便不同使用人员对不同功能的需求，可以根据使用环境、使用目的等个人条件进行安装，自由度更高，给与用户最大的选择权和修改权。

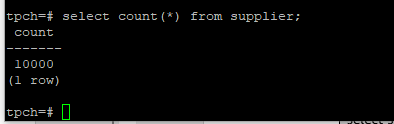
3、使用源码编译时，更容易满足运维、开发人员平时的维护需求，如果二进制打包就可能出现额外的负担，升级组件更方便。

# 关卡二、openGauss数据导入及基本操作

任务一：数据初始化验证

1. 查询supplier表的行数，并将结果进行图：

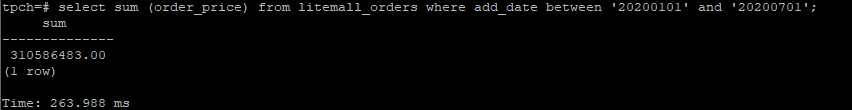
select count(\*) from supplier;;



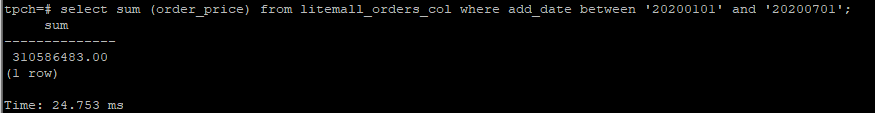
任务二：行存表与列存表执行效率对比

1. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的总和查询，并对比执行效率截图

select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

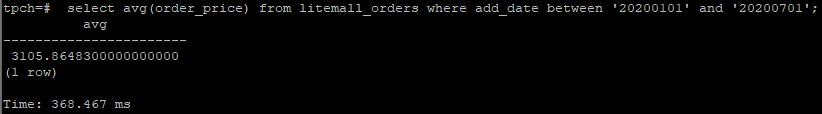


行存表用时：263.988ms

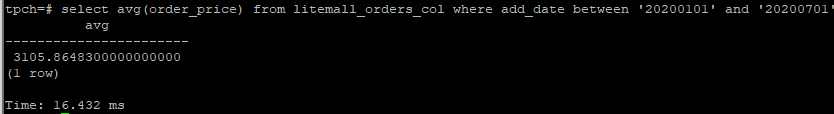
列存表用时：24.753ms

2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

select avg (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

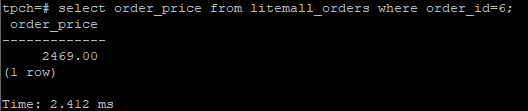


行存表用时：368.467ms

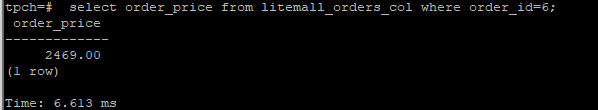
列存表用时：16.432ms

3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

select order\_price from litemall\_orders where order\_id=6;



select order\_price from litemall\_orders\_col where order\_id=6;



行存表用时：2.412ms

列存表用时：6.613ms

4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

update litemall\_orders set order\_price=2468 where order\_id=6;



update litemall\_orders\_col set order\_price=2468 where order\_id=6;



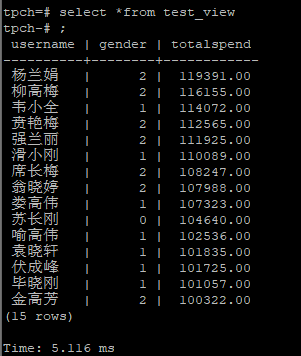
行存表用时：6.304ms

列存表用时：43.413ms

任务三：物化视图的使用

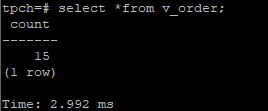
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



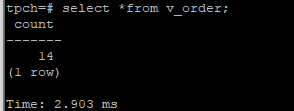
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



实践思考题1：行存表与列存表在执行相同的SQL语句时，为何执行的时间不同？在执行哪些类型SQL时，行存表效率更高？在执行哪些类型SQL时，列存表效率更高？

答：首先，行存表是按照记录一条条被保存在一起，而列存表中，数据被按照一列一列分开保存，在数据读取、写入上存在着差别：行存储写入是一次性完成、消耗的时间较少、能够保存数据的完整性，在数据量大的情况下会影响到数据的处理效率；而列存储在写入效率、保证数据完整性上都不如行存储，但是优势体现在数据的读取，不会产生冗余的数据，在大数据处理领域尤为重要。

行存表效率高：对某一条记录进行增加、修改、删除，查询某一个特定数据对应的同一条记录。

列存表效率高：查询大量记录中的某一列数据，

实践思考题2：全量物化视图与增量物化视图有哪些差别？

答：全量物化视图仅仅只能对创建好的物化视图进行全量更新，不支持增量更新；

增量物化视图可以对物化视图进行增量更新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。

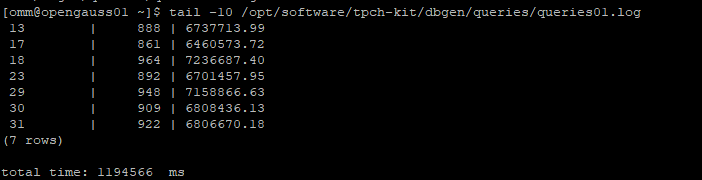
而所谓的全量更新，就是将所有参数传输给后台，进行入库操作；而增量更新，就是传一部分参数然后更新传递的参数。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

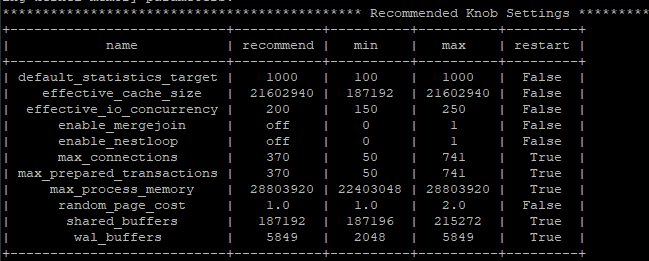
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log



2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

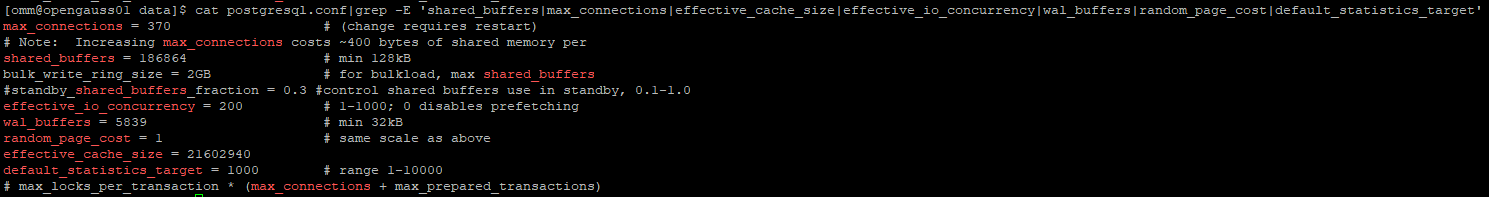
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



3.重启完成后，获取参数值：

cd /opt/software/openGauss/data

cat postgresql.conf|grep -E 'shared\_buffers|max\_connections|effective\_cache\_size|effective\_io\_concurrency|wal\_buffers|random\_page\_cost|default\_statistics\_target'



任务二：使用Index-advisor对select 查询语句进行优化，并通过对比执行计划，得到优化前后的不同。

1. 使用explain，对查询2020年3月订单表收入并进行排序的SQL加以分析，将结果截图。

EXPLAIN

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

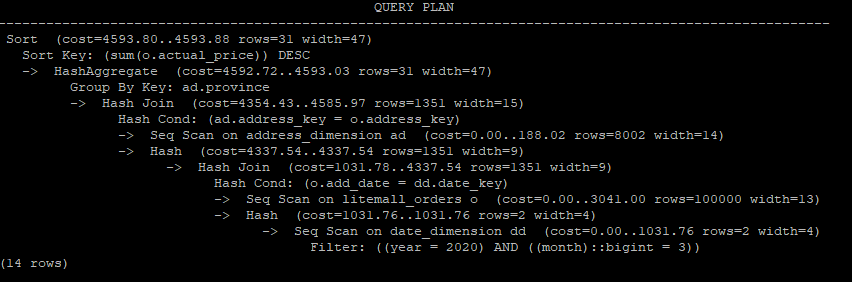
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;



2. 使用索引推荐功能，对查询语句进行推荐，将执行结果截图。

select \* from gs\_index\_advise('

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

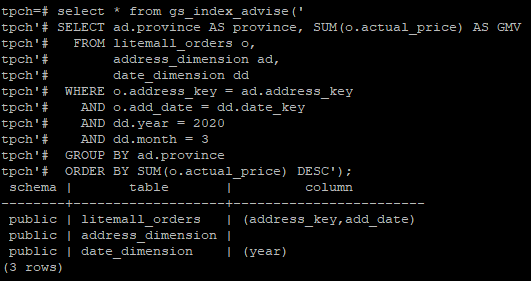
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

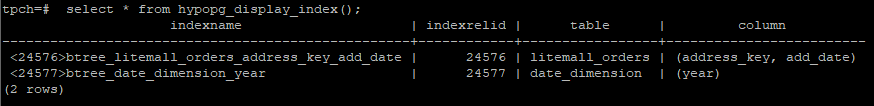
GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC');



3. 查看创建的虚拟索引列，将执行结果截图。

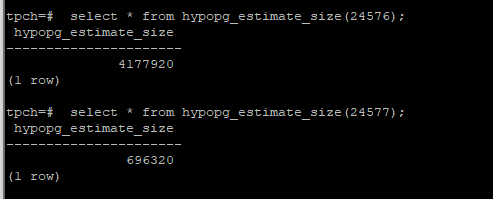
select \* from hypopg\_display\_index();



4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

select \* from hypopg\_estimate\_size(16715);

select \* from hypopg\_estimate\_size(16716);



5.再次使用explain，对该SQL加以分析，将执行结果截图。

EXPLAIN

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

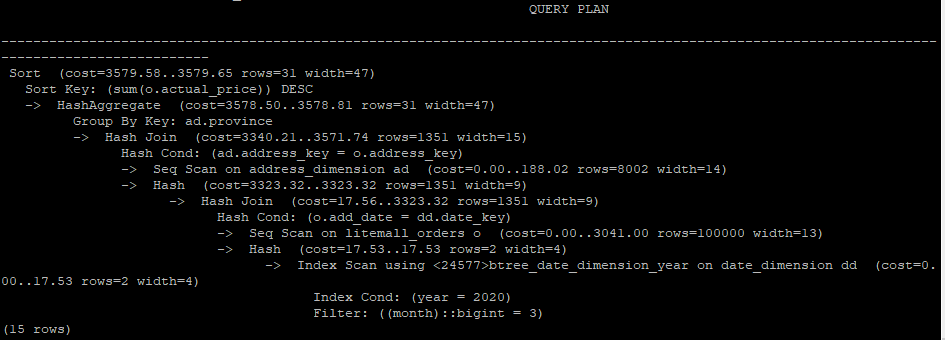
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

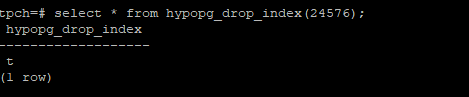
GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;



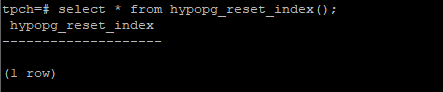
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(16715);



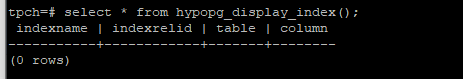
7. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_reset\_index();



8. 查看索引虚拟列，将执行结果截图。

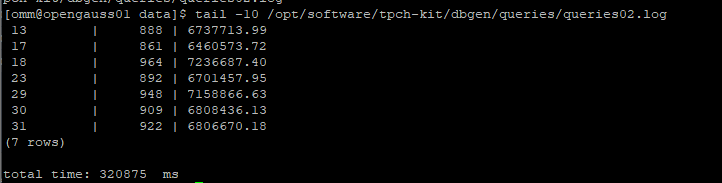
select \* from hypopg\_display\_index();



任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

1. 重新执行queries.sql查询，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

1. 重新执行queries.sql查询，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries03.log

实践思考题1：根据X-Tuner给出的参数优化，在哪些参数上进行了优化，为何要对这些参数进行优化？

答：“default\_statistics\_target=1000”——“默认的统计数据目标”、

“effective\_io\_concurrency=200”——“有效的IO并发”、

“effective\_cache\_size=21602940”——“有效的缓存大小”、

“random\_page\_cost=1”——“随机页面成本”

“wal\_buffers=5839”——“存放wal数据的内存空间”

“shared\_buffers=186864”——“共享缓冲区”

“max\_connections=370”——“最大连接数”

优化数据库参数的目的就是为了更大化地利用数据库资源，充分发挥数据库性能，使得我们使用数据库时，能够得到资源的最大化利用分配。

实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

答：索引能够提高数据库的检索效率，降低IO成本，同时也能降低数据的排序成本，但是在添加索引时，也需要增加一部分的IO和调整索引所计算的量，索引会随着表中数据的憎恨在那个不断增长带来存储空间的消耗。

1、选择最适合的字段属性：对于单独的各个数据的格式属性可以去除冗余。

2、使用连接（join）来代替子查询（Sub-Queries）:减少子查询中需要多个逻辑才能完成的SQL操作，避免事务、表被锁死。

3、使用联合（Union）来代替手动创建的临时表：使用Union来连接多个SQL查询语句。

4、事务：避免语句块中某一条语句失败后，因已经执行语句对于数据库数据完整性、一致性的破坏——事务中语句块要么一起成功、要么就全部失败。Begin….Commit。

5、锁定表：能够维护数据库完整性的事务因为其执行过程中独占使用表的过程，而影响其他用户对于该表的使用，我们可以通过锁定表来确保Locktable后直到UNLocktable时，表中数据不会有其他访客来进行插入、更新、删除。

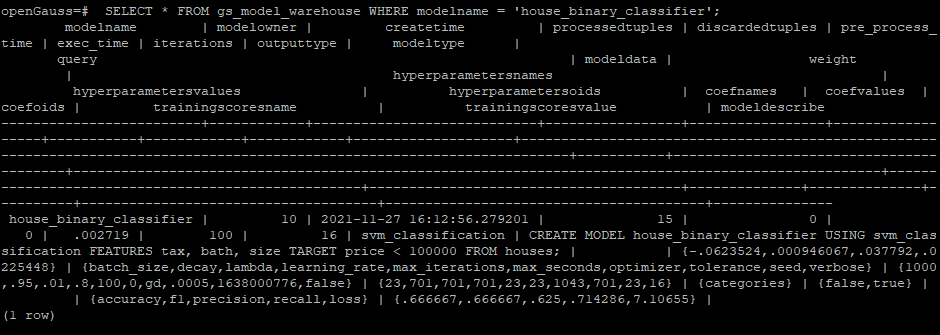
6、使用外键：保证数据的关联性。

7、优化SQL查询语句。

# 关卡四、openGauss的DB4AI特性应用

任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

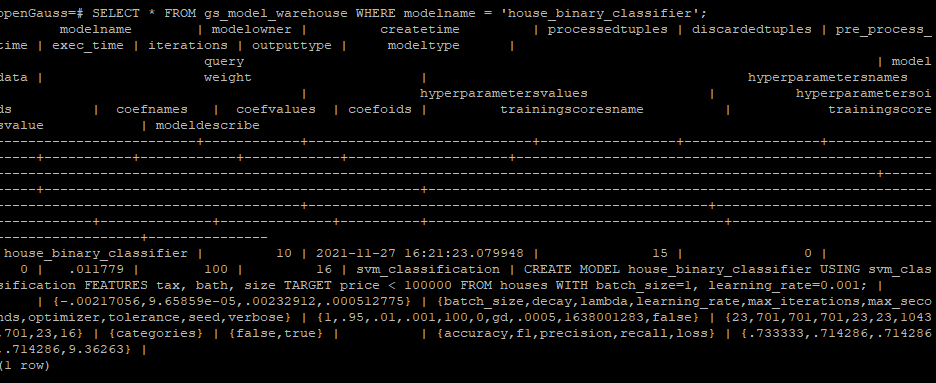
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



预测准确度为：0.66667

任务二：观察新模型的信息，将执行结果截图。

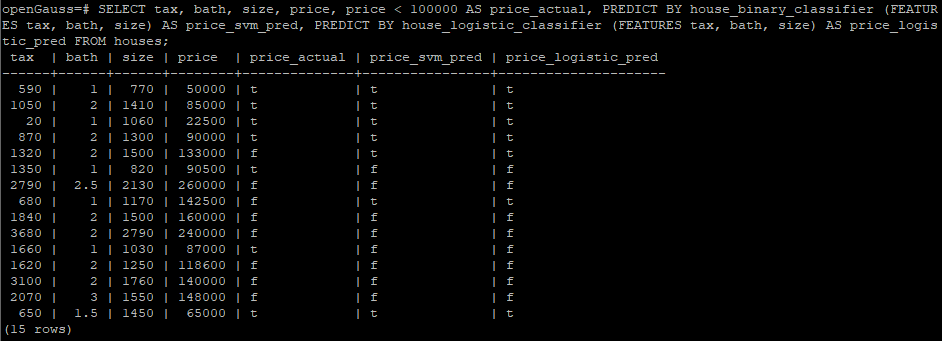
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



预测准确度为：0.73333

任务三：利用训练好的逻辑回归模型预测数据，并与SVM算法进行比较，将执行结果截图。

postgres=# SELECT tax, bath, size, price, price < 100000 AS price\_actual, PREDICT BY house\_binary\_classifier (FEATURES tax, bath, size) AS price\_svm\_pred, PREDICT BY house\_logistic\_classifier (FEATURES tax, bath, size) AS price\_logistic\_pred FROM houses;



实践思考题1：分类模型与回归模型有何不同？

答：分类模型是定性输出，是对离散变量（标签、性质）的预测；回归模型是定量输出，是对连续变量（给出空间）的预测。

实践思考题2：什么是SVM算法？

答：Support Vector Machines，二分类模型，基本模型是定义在特征空间上的间隔最大的线性分类器，学习策略为：使得间隔最大化。

实践思考题3：分类问题有哪些评价指标，请分别说明他们的含义？

答：

准确率（Accurancy）——正确预测样本占总体比例、

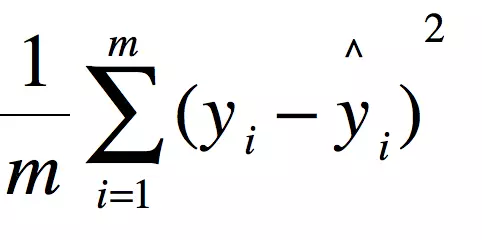
精准率（Precison）——正确且实际为正的样本占预测为正的样本比例、

召回率（Recall）——正确且实际为正的样本占实际为正的样本比例、

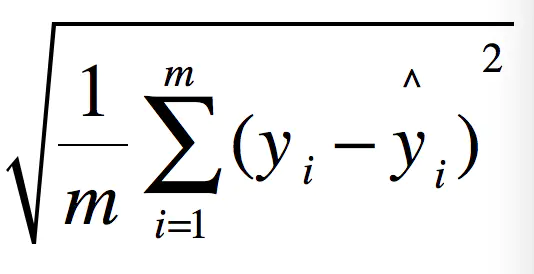
F1-Score——两倍的准确率与召回率之积/准确率与召回率之和

实践思考题4：回归问题有哪些评价指标，请分别说明他们的含义？

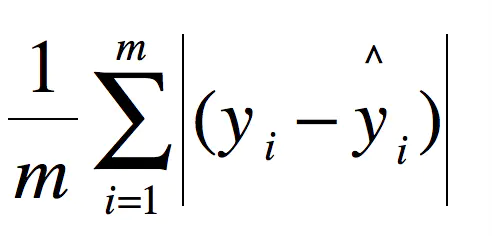
答：均方误差（MSE，Mean Squared Error）,残差的平方求和平均；



均方根误差(RMSE，Root Squard Error)，对均方误差求根



平均绝对误差（MAE，Mean Aboslute Error），残差的绝对值求和平均。



R Squared， 度量拟合优度的一个统计量，又名R方。

