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：

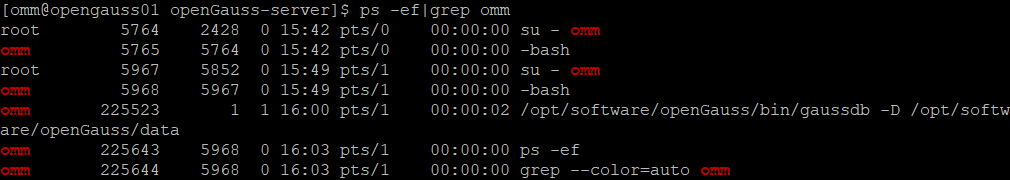
任务一：数据库状态验证

1. 查询数据库状态成功截图

数据库状态

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？

这样可以定制相关参数然后进行安装，安装版本也可以自行进行选择，灵活性大。

源码安装过程中可以设定参数，按照需求进行安装，并且安装的版本，可以自己选择，灵活性比较大。

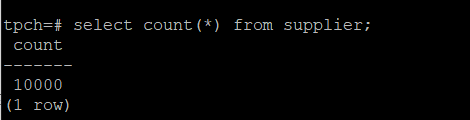
同时可以自行定制，需要什么就定制什么，1对1的满足需求高效。

# 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：

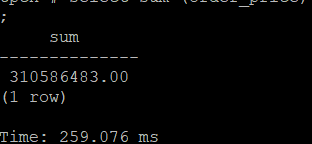
select count(\*) from supplier;;



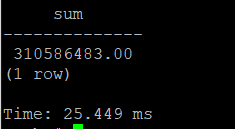
任务二：行存表与列存表执行效率对比

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';

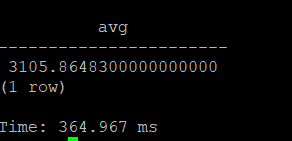


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

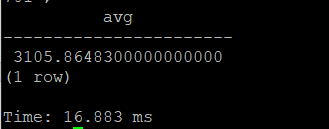


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

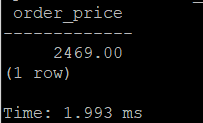


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

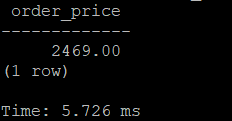


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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



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;



任务三：物化视图的使用

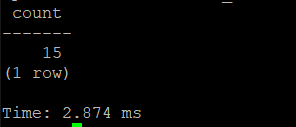
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



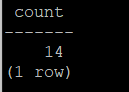
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



6插入数据后刷新物化视图 查询物化试图结果

tpch=# **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？

行存与列存的结构不同。

1.行存储是一行行写入的。而列存储需要把一行记录拆分成单列保存，写入的次数远多于行。再加上磁头需要在盘片上移动和定位花费的时间，实际时间消耗会更大。所以，行存储在写入上占有很大的优势

2. 在修改时，行存储是在指定位置写入一次，列存储是将磁盘定位到多个列上分别写入，所有行存储在数据修改上也是要快于列的。

3. 在读取数据时，行存储通常将一行数据完全读出，如果只需要其中几列数据，就会存在冗余列，而列存储每次读取的数据是集合的一段或者全部，不存在冗余性问题。

4. 由于列储存的数据是同质的，这种情况使数据解析变得容易。行存储则复杂的多，因为在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，这个操作很消耗cpu，所以列存储的解析过程中更有利于分析数据。

行存主要关注某几行的数据，在进行增、删、查、改操作是是比较快的。

列存在求和、求平均值等使用到聚集函数的情境中是比较快的/。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

增量物化视图就是可以对物化视图增量刷新，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。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

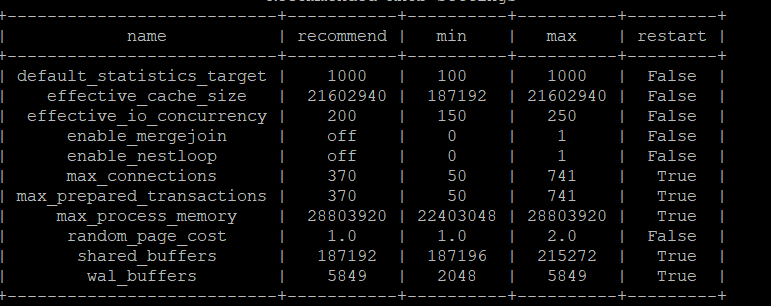
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

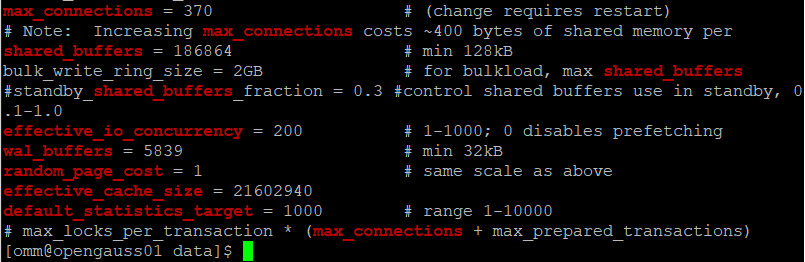
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

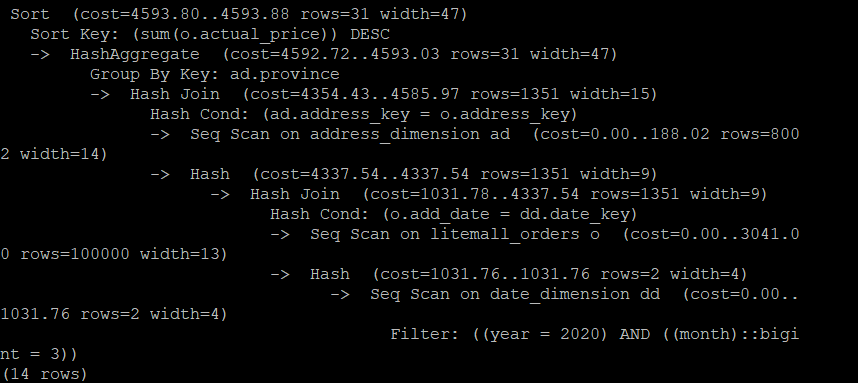
AND o.add\_date = dd.date\_key

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

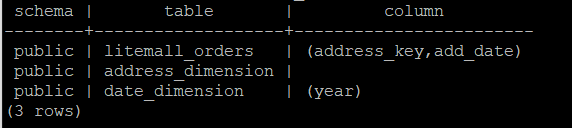
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

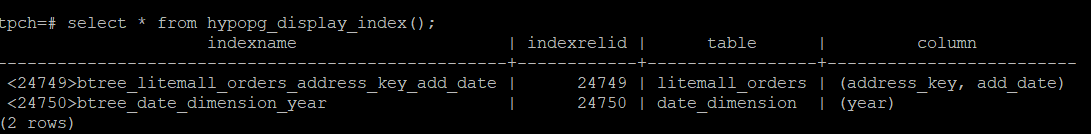
GROUP BY ad.province

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。

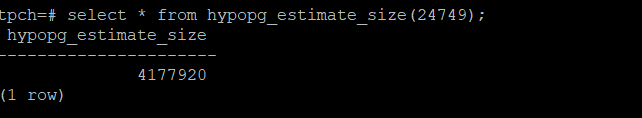
select \* from hypopg\_display\_index();

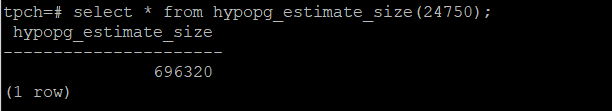


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

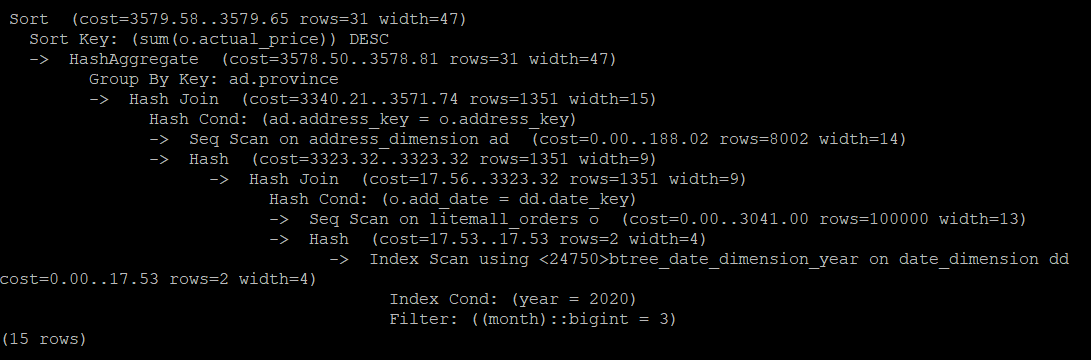
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

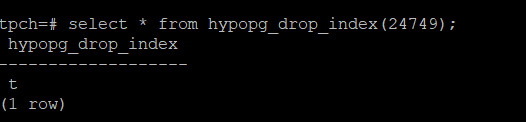
GROUP BY ad.province

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



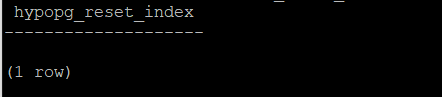
6. 删除某一个索引虚拟列，将执行结果截图。

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



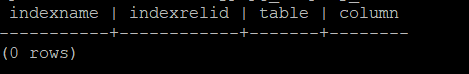
7. 删除所有索引虚拟列，将执行结果截图。

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？

shared\_buffers

max\_connections

effective\_cache\_size

effective\_io\_concurrency

wal\_buffers

random\_page\_cost

default\_statistics\_target

shared\_buffer是数据页缓冲区数据库在查询前，会先查找shared\_buffer的页，如果命中，就直接返回，避免从磁盘中查询。类似于快表机制。

max\_connections参数用来设置最大连接（用户）数，该参数在服务器资源够用的情况下应该尽量设置大，以满足多个客户端同时连接的需求。

effecve\_cache\_size这个参数只在查询优化器选择时使用，并不是实际分配的内存，该参数越大，查询优化器越倾向于选择索引扫描。

存取数据需要操作io，会很大程度上影响效率，所以也需要优化

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？

1. 通过唯一性索引（unique）可确保数据的唯一性；
2. 加快数据的检索速度，加快表之间的连接
3. 减少分组和排序时间
4. 使用优化隐藏器提高系统性能

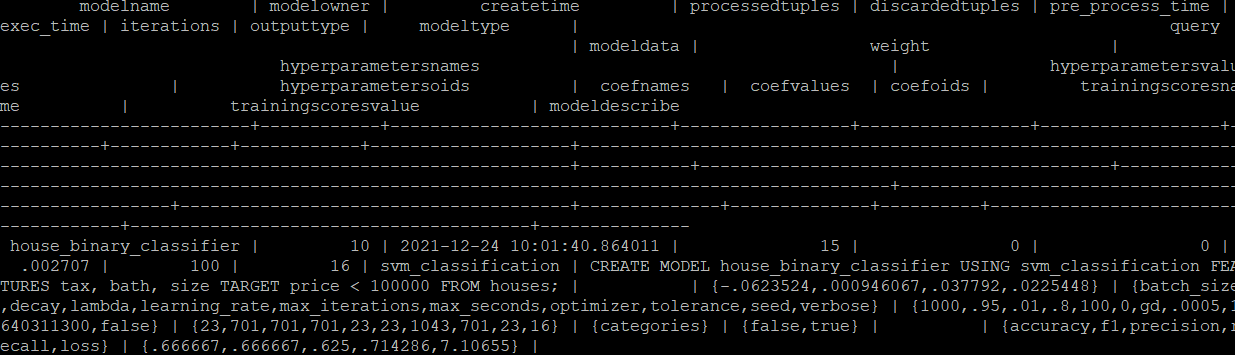
调整服务器的内存分配，调整硬盘I/O，调整操作系统参数。

优化字段类型，数据库最耗时的就是IO处理，所以尽可能减少IO读写量。

# 关卡四、openGauss的DB4AI特性应用

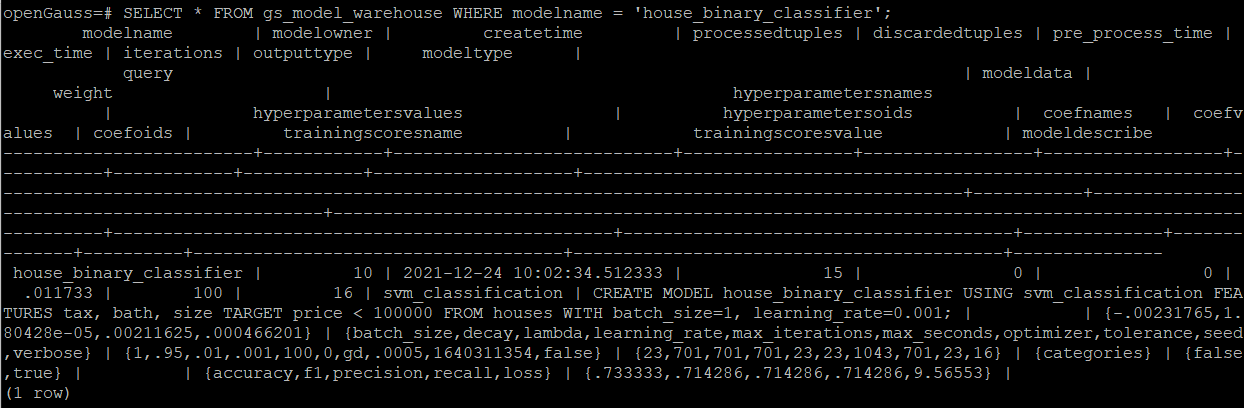
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



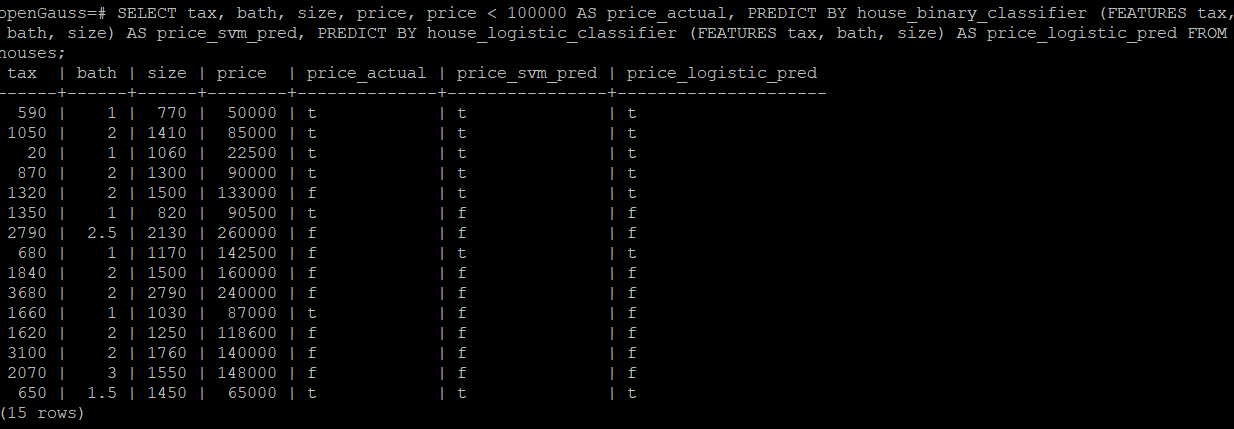
任务二：观察新模型的信息，将执行结果截图。

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



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？

分类和回归的区别在于输出变量的类型。

定量输出称为回归，或者说是连续变量预测；

定性输出称为分类，或者说是离散变量预测。

实践思考题2：什么是SVM算法？

SVM的全称是Support Vector Machine，即支持向量机，主要用于解决数据分类问题，属于有监督学习算法的一种，它被认为是监督学习的最佳算法之一。 该算法的主要思想是将数据从维数相对较低的空间映射到维数相对较高的空间，从而通过超平面将较高维的数据分为两类。 以最大间隔分隔数据的超平面称为支持向量分类器，这个超平面可以使用核函数确定。

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？

Accuracy：表示预测模型的精确度,是预测模型预测正确的个数占数据集中实例总个数的比例。

Precision：表示查准率,是预测模型预测为有缺陷实例中真实类别为有缺陷的数据所占的比例。

Recall：表示召回率,是预测模型预测为有缺陷的实例的数量占真实有缺陷的实例数的比例。

F-measure：是一个综合评价指标,提供了召回率和精确度之间的权衡。

G-mean：几何平均数，评价类不平衡数据的表现.对于软件缺陷数据来说,有缺陷的模块占少数,无缺 陷的模块占多数,所以有类不平衡的问题。

AUC：表示 ROC 曲线下的面积.ROC 曲线叫作接受者工作特征曲线,其横坐标为假阳性率,纵坐标为 Recall.一般情况下,AUC 值越大,说明缺陷预测模型的性能越好。

MCC：表示实际分类与预测分类之间的相关系数.该指标同时考虑了 TP、FN、FP 及 TN,是一个相对均衡的指标,能够评价类不平衡数据的表现。

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？

RMSE（Root Mean Square Error）均方根误差衡量观测值与真实值之间的偏差，常用来作为机器学习模型预测结果衡量的标准。

MSE（Mean Square Error）均方误差是真实值与预测值的差值的平方然后求和平均。通过平方的形式便于求导，所以常被用作线性回归的损失函数。

MAE（Mean Absolute Error）平均绝对误差是绝对误差的平均值。可以更好地反映预测值误差的实际情况。

SD（Standard Deviation）标准差方差的算术平均根。用于衡量一组数值的离散程度。

MASE（Mean Scaled Percentage Error）平均平方百分比误差，即求所有样本的真实值与预测值的差与真实值的比例求平方的和求平均。

MAPE(Mean Absolute Percentage Error)即求所有样本真实值与预测值差绝对值与真实值的比例的和求平均。

R平方值是决定系数，反应因变量的全部变异能通过回归关系被自变量解释的比例。