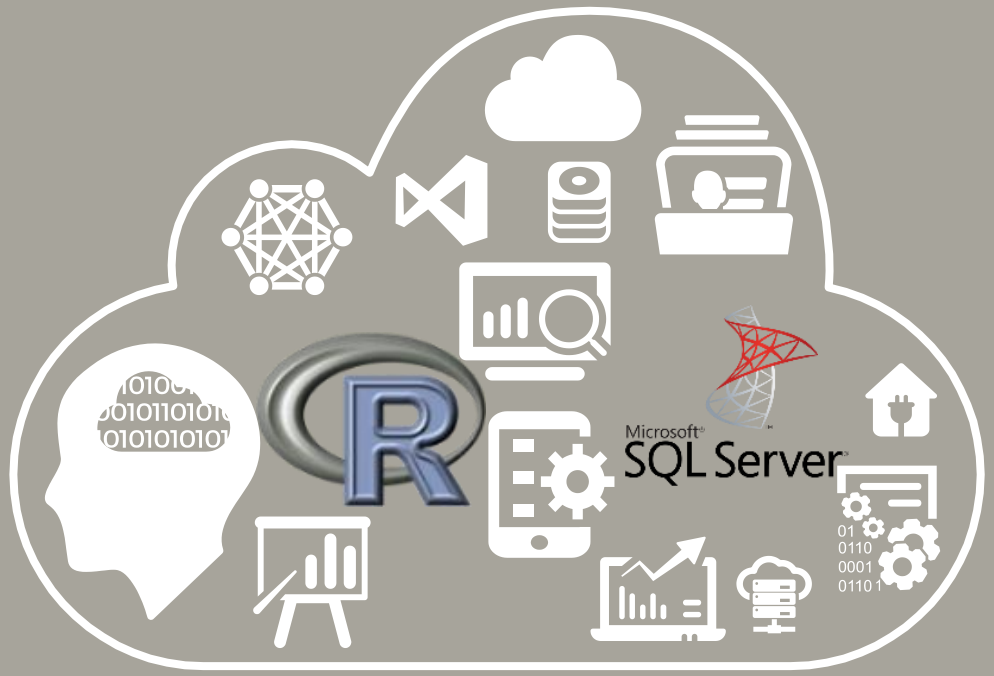
www.pwc.com

Być bliżej danych, czyli R + SQL Server 2016

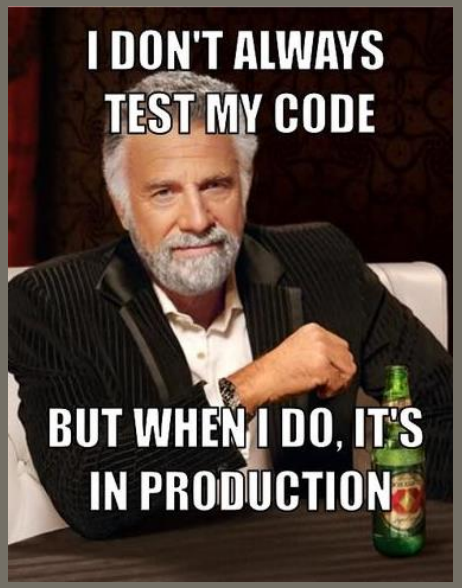
Piotr Bochnia, SER19



pwc

Agenda

1. Krótko o mnie
2. Dlaczego SQL Server?
3. Jak połączyć R i SQL Server? – podejście tradycyjne
4. Jak lepiej połączyć R i SQL Server? – SQL Server R Services
5. Przykłady nowych funkcjonalności
6. Ciekawostki



Krótko o mnie...

Edukacja

- ***MIM UW:*** *Matematyka*
- ***SGH:*** *Metody Ilościowe*

Zawodowo

- ***Data Scientist/konsultant*** *w zespole Data Analytics w PwC*
- *Projekty głównie dla branży* ***bankowej*** *i* ***retail***
- ***R*** *jako główne narzędzie do tworzenia modeli*

Kontakt

- *piotr.bochnia@pl.pwc.com*
- *pl.linkedin.com/in/PiotrBochnia*

Dlaczego SQL Server?

SQL Server jest popularną bazą danych umożliwiającą wydajne przechowywanie i przetwarzanie danych

Microsoft SQL Server

- *System zarządzania bazą danych (database management system, DBMS)*
- *Umożliwia efektywne przechowywanie i udostępnianie danych*
- *Przetwarzanie danych – język Transact-SQL (T-SQL)*
- *Wysoka wydajność*
- *Popularne narzędzie*

Ale...

Brak bardziej zaawansowanej analityki (nie licząc SSAS)

Problemy z wizualizacją danych (nie licząc SSRS)

R wydaje się być doskonałym uzupełnieniem SQLa w zakresie zaawansowanej analityki, jednak istnieje kilka problemów

- *Zaawansowana statystyka*
- *modele*
- *wizualizacje*

Łatwo rozszerzalne funkcjonalności

Rozwinięta społeczność użytkowników

Ale…

Dane ładowane (zazwyczaj) do pamięci

Problemy z wydajnością (jednowątkowość)

Brak komercyjnego wsparcia

Jak to połączyć?

Tradycyjnie wykorzystanie funkcjonalności R na danych z bazy SQL oznaczało konieczność kopiowania danych do pamięci lokalnego komputera

Tradycyjny scenariusz pracy w R na danych z bazy SQL Server

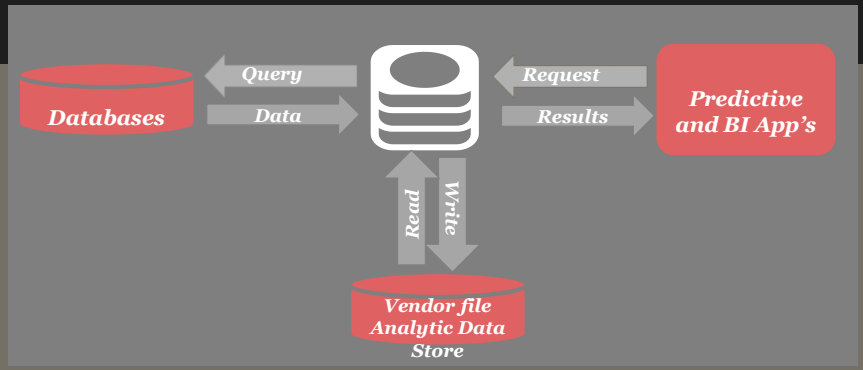
```
library(RODBC)

connStr <- "Driver=SQL Server;Server=13.79.174.91;Database=Titanic;Uid=RUser;Pwd=1Qazwsx."
query <- "SELECT PassengerId, Survived, Pclass, Sex, Age, SibSp, Parch, Fare ,Embarked FROM Titanic.dbo.DataModelTrain"


odbcConnection <- odbcDriverConnect(connection =  connStr)

train.data <- sqlQuery(channel = odbcConnection, query = query)


logit <- glm(formula = Survived ~ . - PassengerId, family = binomial(link = logit), data = train.data)

predictions <- predict(logit, train.data)
predictions.data <- data.frame(PassengerId = train.data$PassengerId, prediction = predictions)

sqlSave(channel = odbcConnection, dat = predictions.data, tablename = "dbo.TrainLogitPredictions")

odbcClose(channel = odbcConnection)
```

1 *Połączenie z bazą danych SQL Server (np. RODBC)*

2 *Ściąganie danych z bazy do data frame'u*

3 *Budowa modelu w R*

4 *Zapisanie wyników do bazy*



Klasyczne podejście wymaga przenoszenia danych oraz cechuje je niska wydajność i trudności z operacjonalizacją

Wady tradycyjnego podejścia

1 ***Konieczność kopiowania danych***

- *Mało wydajne, zwłaszcza gdy danych jest dużo*
- *Mało bezpiecznie*

2 ***Słaba skalowalność / wydajność***

- *Jednowątkowość*
- *Dane muszą mieścić się w pamięci*

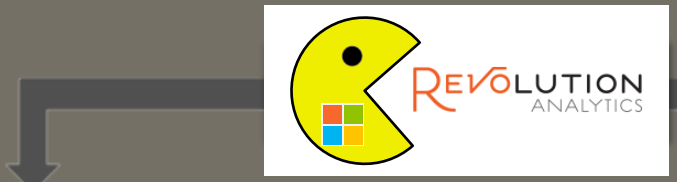
3 ***Problemy z przejściem na produkcję***

- *Jak odpalać skrypty R z poziomu aplikacji?*
- *Często: konieczność przepisania R na inny język*

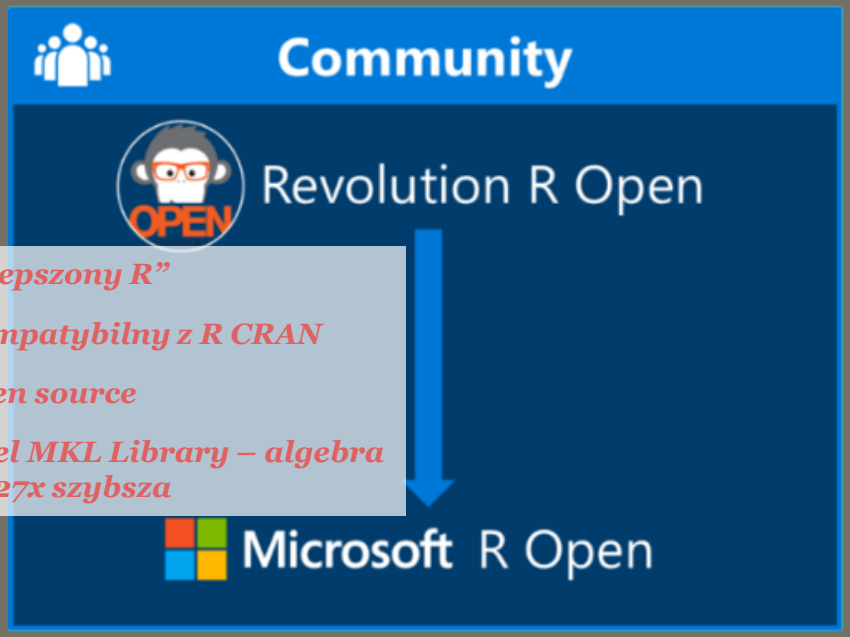
Jak to połączyć lepiej?

SQL Server R Services jest nową platformą, która umożliwia uruchamianie wydajnego kodu R w środowisku SQL Servera 2016…

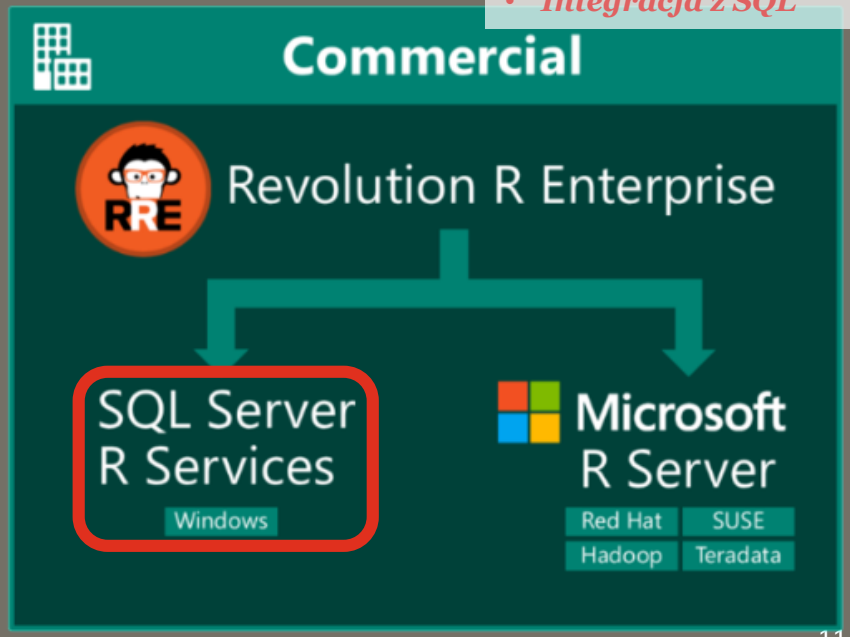
- Kwiecień 2015 – Przejęcie Revolution Analytics przez Microsoft
- Maj 2015 – pierwsza wersja CTP SQL Server 2016
- Czerwiec 2016 – oficjany release



- *Pakiet ScaleR – wysoko wydajna statystyka i data mining*
- *Komercyjne wsparcie*
- *Integracja z SQL*



- *„Ulepszony R”*
- *Kompatybilny z R CRAN*
- *Open source*
- *Intel MKL Library – algebra do 27x szybsza*



… oraz adresuje wspomniane wcześniej problemy z tradycyjnym modelem korzystania z R + SQL Server

Analityka w środowisku bazy danych (In-Database analytics)

- *Możliwość uruchamiania skryptów R w środowisku SQL Servera – **konteksty obliczeniowe***
- ***Brak konieczności kopiowania** danych na maszynę kliencką*

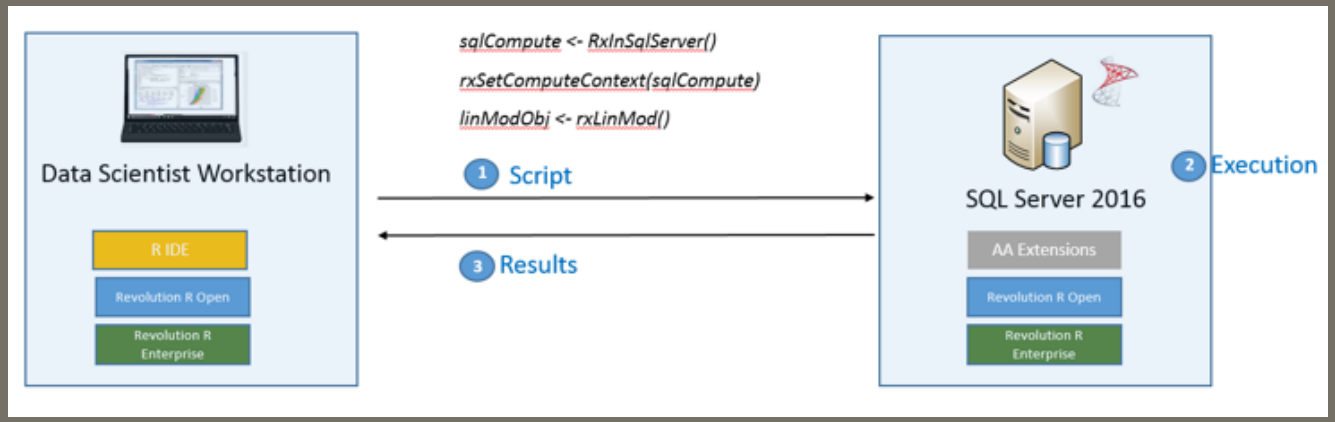
Operacjonalizacja skryptów R

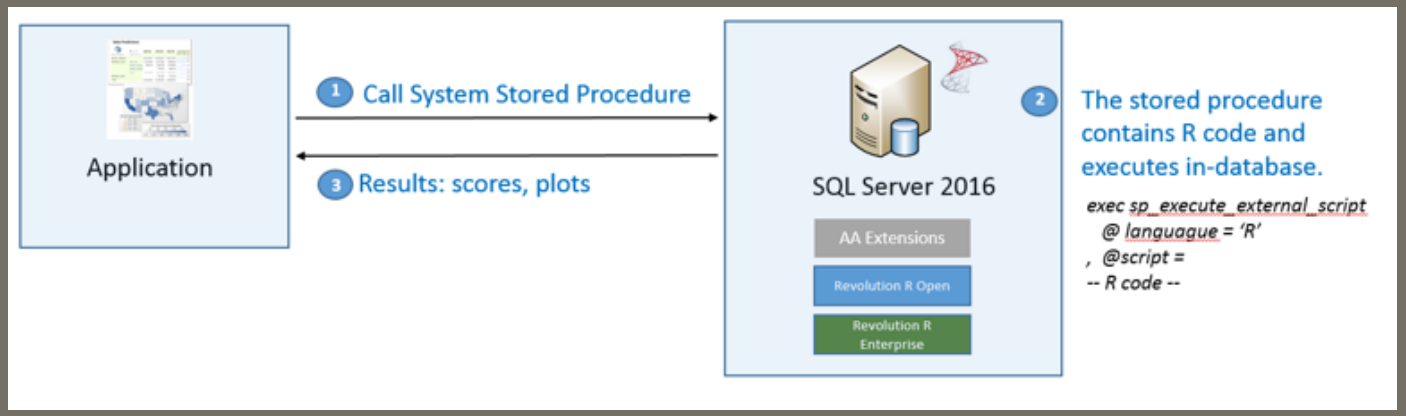
- *Możliwość zagnieżdżania **kodu R w kodzie SQL***
- *Uruchamianie skryptów R przez zewnętrzne aplikacje za pomocą **procedur języka T-SQL** (stored procedures)*

Wydajność / skalowalność

- *Wykorzystanie* pakietu ***RevoScaleR*** z Microsoft R Server
- Wielowątkowość i zrównoleglanie obliczeń
- Brak ograniczeń *pamięciowych*

Kluczową zaletą SQL R Services jest przeniesienie ciężaru obliczeń analitycznych bliżej danych, czyli do SQL Servera, co widać w dwóch typowych scenariuszach





Jak to połączyć lepiej? - przykład

Ustawienie kontekstu obliczeniowego na bazę danych pozwala na trenowanie modelu i scorowanie po stronie serwera, bez zbędnego przenoszenia danych

R + SQL Server z R Services

```
connStr <- "Driver=SQL Server;Server=13.70.200.187;Database=Titanic;Uid=RUser;Pwd=1Qazwsx."
query <- "SELECT PassengerId, Survived, Pclass, Sex, Age, SibSp, Parch, Fare ,Embarked FROM Titanic.dbo.DataModelTrain"

dbContext <- RxInSqlServer(connectionString = connStr)
rxSetComputeContext(dbContext)

train.data <- RxSqlServerData(sqlQuery = query, connectionString = connStr,
                              stringsAsFactors = TRUE)

logit <- rxLogit(formula =
                 Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
                 data = train.data)


predictions <- RxSqlServerData(connectionString = connStr,
                               table = "dbo.TrainRxLogitPredictions")

rxPredict(modelObject = logit, data = train.data, outData = predictions,
          extraVarsToWrite = c("PassengerId", "Survived"), predVarNames = "prediction",
          type = "response", overwrite = TRUE)
```

1. *Ustawienie kontekstu obliczeniowego na bazę danych*
2. *Stworzenie referencji do danych na serwerze (bez kopiowania!)*
3. *Budowa modelu w RevoScaleR – po stronie SQL*
4. *Scorowanie po stronie SQL i zapis scorów do tabeli w bazie*

Pakiet RevoScaleR oferuje spory zbiór wydajnie zaimplementowanych algorytmów statystycznych i data minigowych

Scale R – Parallelized Algorithms & Functions

Data Preparation

- Data import – Delimited, Fixed, SAS, SPSS, OBDC
- Variable creation & transformation
- Recode variables
- Factor variables
- Missing value handling
- Sort, Merge, Split
- Aggregate by category (means, sums)

Descriptive Statistics

- Min / Max, Mean, Median (approx.)
- Quantiles (approx.)
- Standard Deviation
- Variance
- Correlation
- Covariance
- Sum of Squares (cross product matrix for set variables)
- Pairwise Cross tabs
- Risk Ratio & Odds Ratio
- Cross-Tabulation of Data (standard tables & long

Statistical Tests

- Chi Square Test
- Kendall Rank Correlation
- Fisher's Exact Test
- Student's t-Test

Sampling

- Subsample (observations & variables)
- Random Sampling

Predictive Models

- Sum of Squares (cross product matrix for set variables)
- Multiple Linear Regression
- Generalized Linear Models (GLM) exponential family distributions: binomial, Gaussian, inverse Gaussian, Poisson, Tweedie. Standard link functions: cauchit, identity, log, logit, probit. User defined distributions & link functions.
- Covariance & Correlation Matrices
- Logistic Regression
- Classification & Regression Trees

Variable Selection

- Stepwise Regression

Simulation

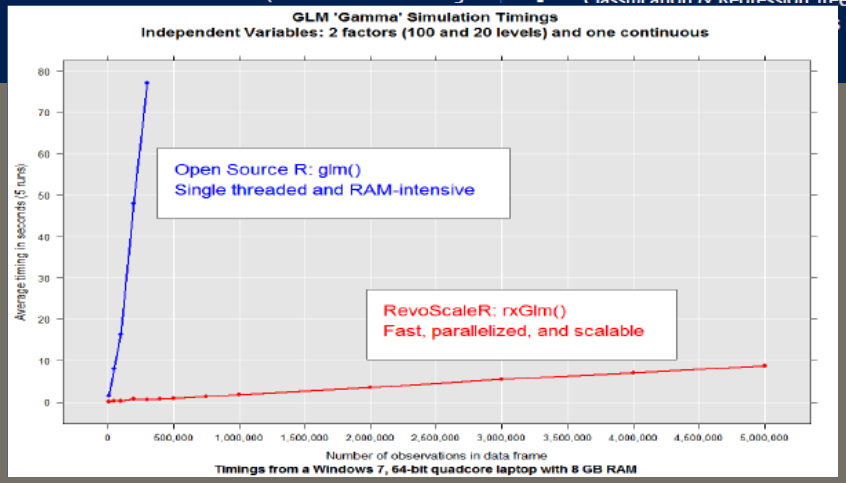
- Simulation (e.g. Monte Carlo)
- Parallel Random Number Generation

Cluster Analysis

- K-Means

Classification

- Decision Trees
- Decision Forest
- Gradient Boosted Decision Trees
- Naïve Bayes



File Name	Compressed File Size (MB)	No. Rows	Open Source R (secs)	Revolution R (secs)
Tiny	*0.3*	*1,235*	*0.00*	*0.05*
V. Small	*0.4*	*12,353*	*0.21*	*0.05*
Small	*1.3*	*123,534*	*0.03*	*0.03*
Medium	*10.7*	*1,235,349*	*1.94*	*0,08*
Large	*104.5*	*12,353,496*	*60.69*	*0.42*
Big (full)	*12,960.0*	*123,534,969*	*Memory!*	*4.89*
V. Big	*25,919.7*	*247,069,938*	*Memory!*	*9.49*
Huge	*51,840.2*	*494,139,876*	*Memory!*	*18.92*

W przypadku brakujących funkcjonalności w RevoScaleR, możliwe jest korzystanie z zewnętrznych pakietów R

Korzystanie z innych pakietów – funkcja rxExec

```
trainXGBoost <- function(train.data) {
    library(xgboost)
    library(dplyr)

    train.df <- rxImport(train.data)

    train.mat <- train.df %>% select( - PassengerId, - Survived) %>% data.matrix
    response <- train.df$Survived

    xgb <- xgboost(data = train.mat, label = response, nrounds = 2,
                   objective = "binary:logistic", missing = NaN)
    xgb
}

xgb <- rxExec(FUN = trainXGBoost, train.data)
```

1. *Ładujemy zewnętrzne pakiety*
2. *To pobiera dane do pamięci, ALE: to będzie wykonywane na maszynie SQL Servera*
3. *Trenujemy model i zwracamy go*
4. *Wywołanie po stronie serwera poprzez funckję rxExec (bo mamy dobrze ustawiony computeContext!)*

Połączenie R z bazą danych pozwala wykorzystać ją jako repozytorium modeli/obiektów R, które mogą być tam przechowywane w polach tabel w zserializowanej postaci

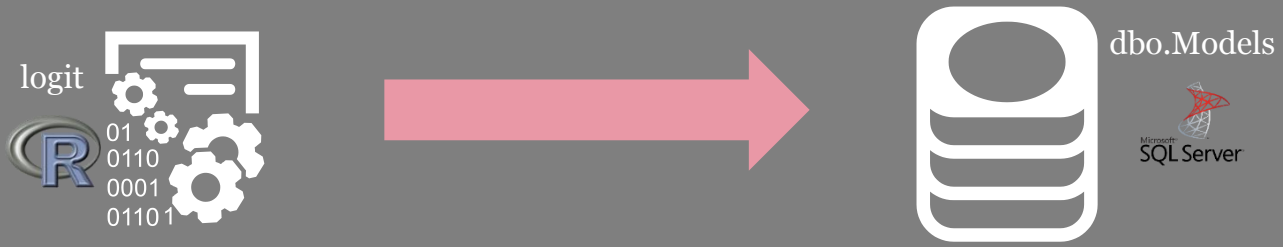
Serializacja i przechowywanie obiektów R w SQL Server

```
persistModel <- function(model, description) {
    model.serialized <- model %>% serialize(NULL) %>% paste(collapse = "")
    model.data <- data.frame(model = model.serialized, description = description)
    library(RODBC)
    odbcConnection <- odbcDriverConnect(connection = connStr)
    query <- paste0("INSERT INTO dbo.Models (model, description) VALUES( CONVERT(VARBINARY(MAX), '",
                    model.serialized, "', 2), '", description, "')")
    sqlQuery(odbcConnection, query)
    odbcClose(odbcConnection)
}

persistModel(logit, "Logistic regression model")
```

1 *Serializacja modelu do formatu binarnego*

2 *Zapisanie zserializowanego modelu do bazy danych*

Model po serializacji zapisywany jest w tabeli dbo.Models w polu typu VARBINARY(MAX)



Nowa procedura T-SQL do uruchamiania skryptów R umożliwia ich łatwiejszą operacjonalizację

Konsumowanie zapisanego modelu przy użyciu sp_execute_external_script

```
CREATE PROCEDURE [dbo].[ScoreWithLatestModel] @inquery nvarchar(max)
AS
BEGIN

  DECLARE @model varbinary(max) = (SELECT TOP 1 model FROM dbo.Models ORDER BY id DESC);
  EXEC sp_execute_external_script @language = N'R',
                                  @script = N'
       mod <- unserialize(as.raw(model));
       print(summary(mod))
       OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL,
                 predVarNames = "Score", type = "response",
                  overwrite = TRUE, checkFactorLevels=FALSE, extraVarsToWrite = c("PassengerId", "Survived"));
       str(OutputDataSet)
       print(OutputDataSet)',
                                  @input_data_1 = @inquery,
                                  @params = N'@model varbinary(max)',
                                  @model = @model
  WITH RESULT SETS ((Score float, PassengerId int, Survived int));

END
```

1. *Definicja procedury, parametry*
2. *Konsumujemy wcześniej zapisany model z tabeli dbo.Models*
3. *Nowa procedura sp_execute_external_script* – do uruchamiania skryptów R
4. *Zagnieżdżony kod R*
5. *Procedura zwraca zbiór danych: score, id pasażera i prawdziwa wartość zmiennej zależnej*

Wywołanie procedury i przechwytywanie wyników z dowolnej aplikacji

```
EXEC dbo.ScoreWithLatestModel @inquery = 'SELECT * FROM Titanic.dbo.DataModelTrain'
```

Podsumowanie

Wadami tradycyjnego modelu współpracy Ra z SQLem są konieczność kopiowania danych, słaba wydajność oraz problemy z operacjonalizacją skryptów

SQL Server R Services pozwala wykorzystać zalety R jako narzędzia analitycznego oraz SQL jako platformy do przechowywania danych

Kluczowe funkcjonalności

Analityka w bazie - Konteksty obliczeniowe (w tym SQL Server Compute Context)

R w procedurach T-SQl

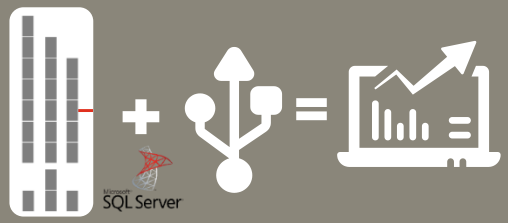
Pakiet RevoScaleR

Potencjał rozwojowy

Varia

Wykorzystanie struktur SQL:

klucze, columnstore index + paralelizacja = wysoka wydajność



Inne konteksty obliczeniowe: **local parallel, Hadoop, Spark** (R Server on HDInsight **#bigdata**)

XDF – gdy dane nie mieszczą się w pamięci

Pakiet checkpoint - Latest Isn't Always Greatest

```
checkpoint("2016-06-08")
```

R Tools for Visual Studio – alternatywa dla **RStudio**?

Q&A