



WOMEN IN DATA SCIENCE



#WiDS2020

¿Cómo organizar tus experimentos de ML? MLFlow, tu mejor amiga.



Melina Solovey

Data Scientist, Pi Data Strategy & Consulting





Colonia Caroya





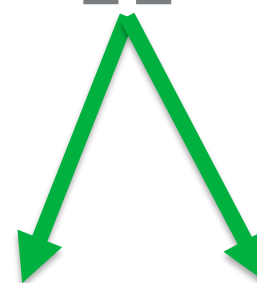
SEGA®



PC
GAMER



WOMEN IN DATA SCIENCE



? 1 2



WOMEN IN DATA SCIENCE



WOMEN IN DATA SCIENCE

¿Cómo organizar tus experimentos de ML?
MLFlow, tu mejor amiga.



Objetivo

Conocer una herramienta que nos facilita el trabajo de administrar el ciclo de vida de ML, incluyendo la experimentación, reproducibilidad y deploy de modelos, además de brindar la posibilidad de comparar la performance de los modelos obtenidos a lo largo del proyecto.



Contenidos

- Recordando el método científico
- Escenario Tecnológico
- MLFlow:
 - Introducción de la herramienta
 - Proceso
 - Notebook con ejemplo



Método Científico

Planteamiento
del Problema



Observación



Hipótesis

Experimentación

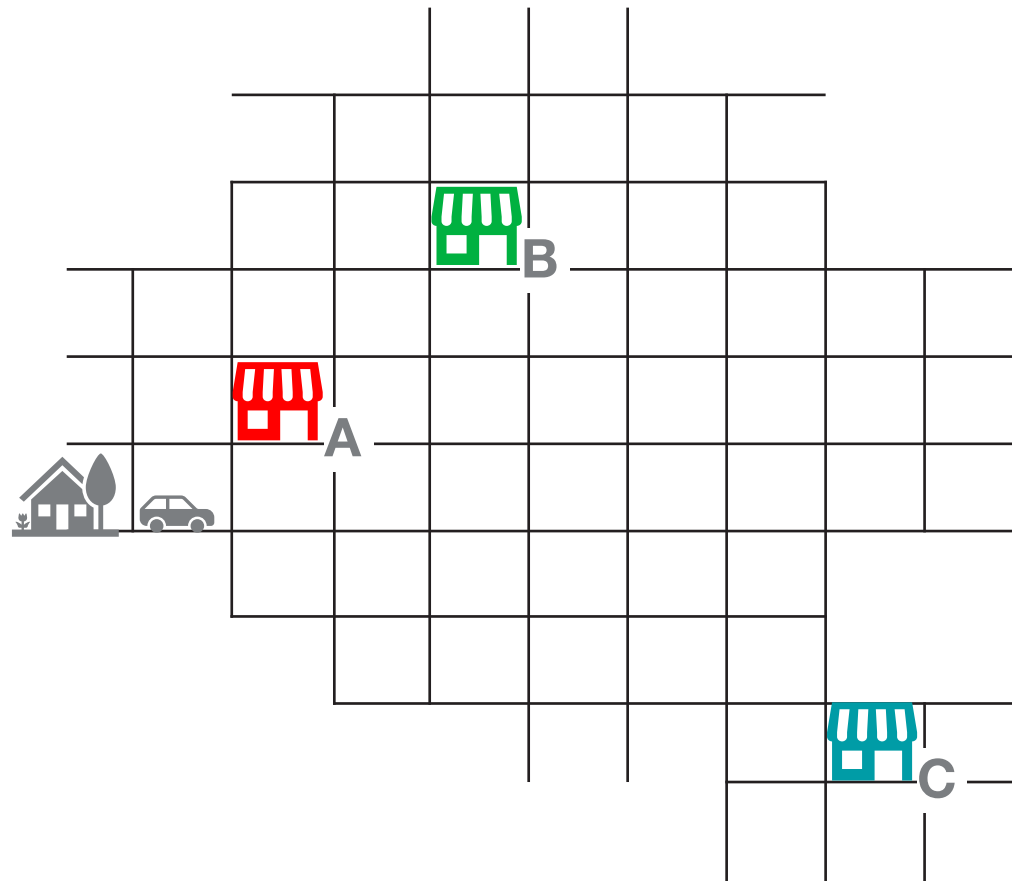


Análisis de
Resultados

Conclusión



Método Científico



Distancia

Método Científico

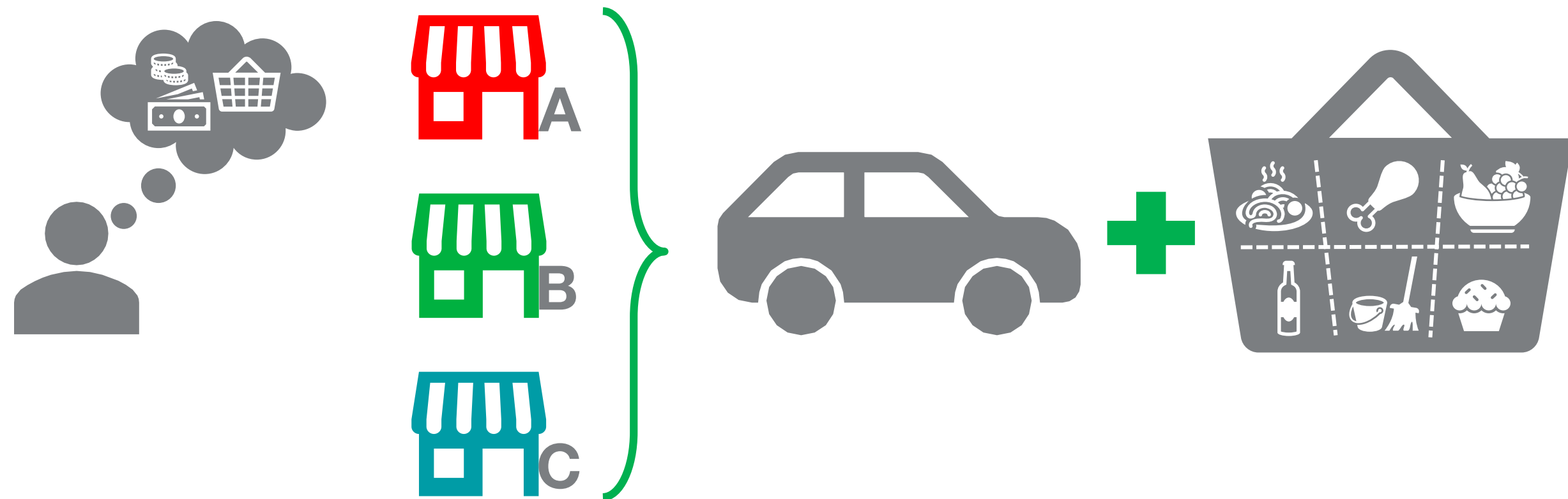
costo_supermercado



Problema: ¿A qué super me conviene ir hacer las compras?

Nombre del experimento:

costo_supermercado



Método Científico

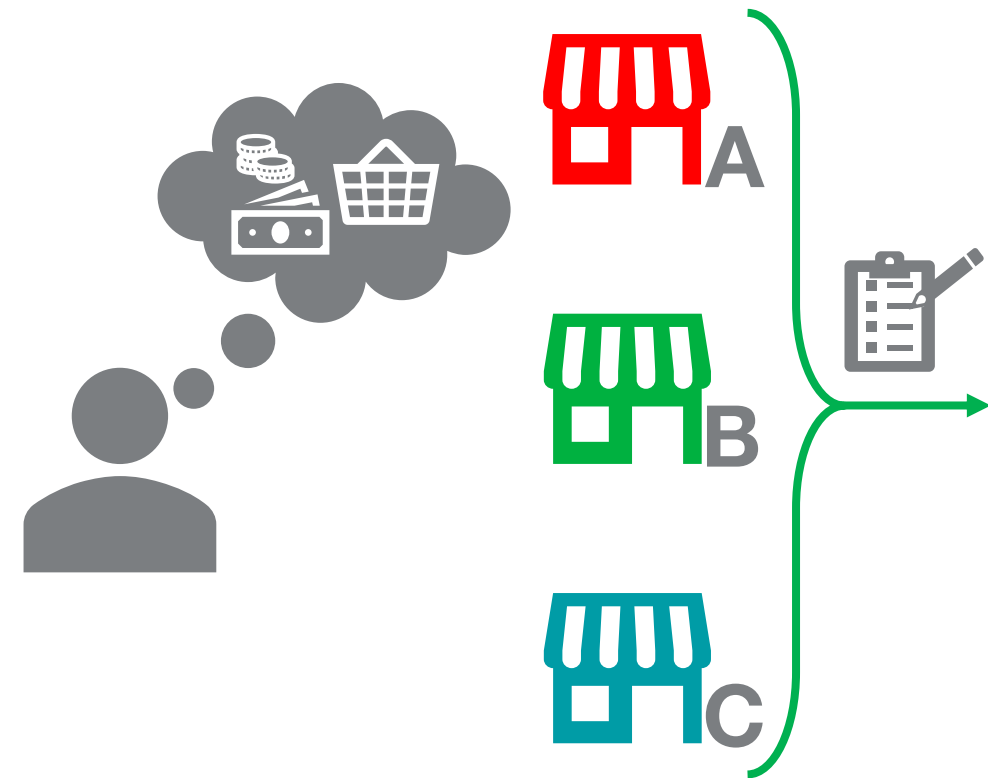
costo_supermercado






WOMEN IN DATA SCIENCE

Método Científico

costo_supermercado



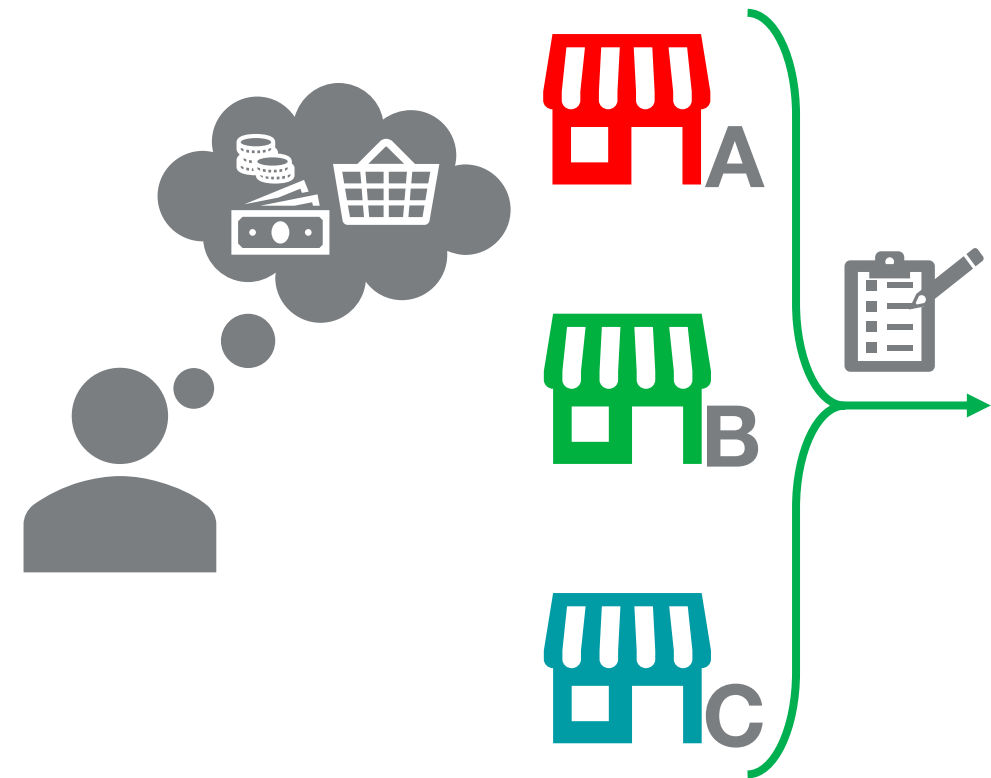
05/08/2019

								
1	A	100	220	90	90	110	50	0
2	B	110	180	100	90	150	45	50
3	C	90	170	90	100	90	40	100



Método Científico

costo_supermercado



05/08/2019

1	A	100	220	90	90	110	50	0	660
2	B	110	180	100	90	150	45	50	725
3	C	90	170	90	100	90	40	100	680



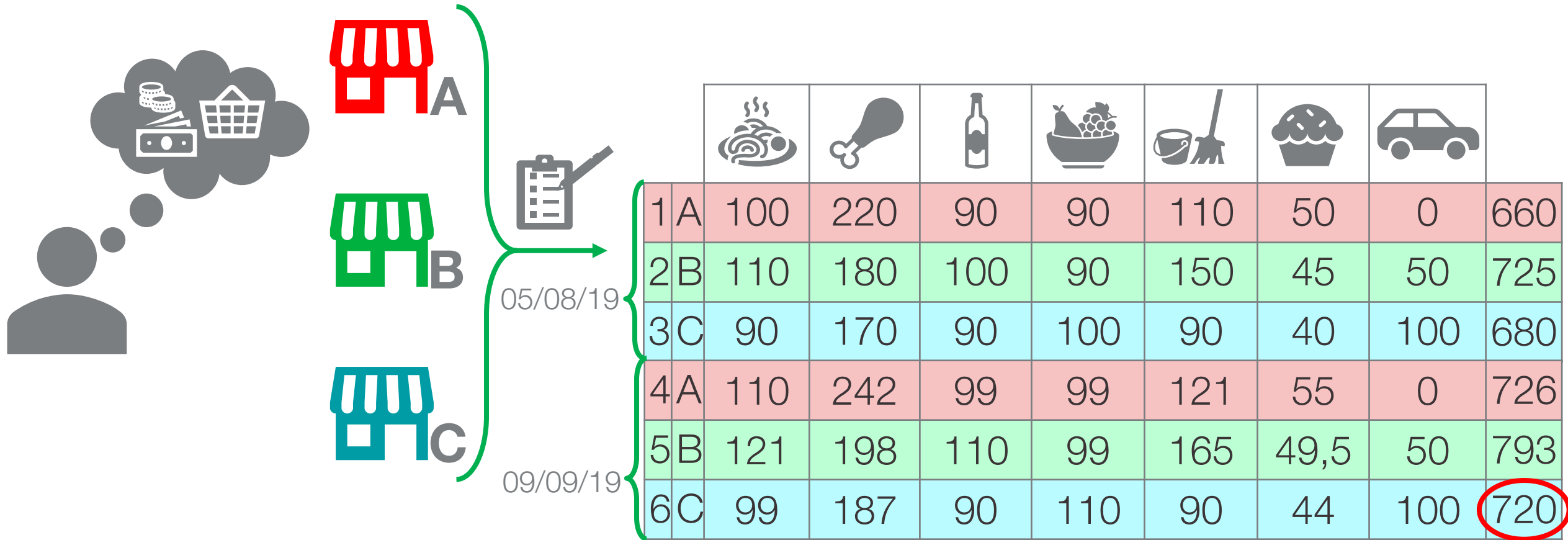
Método Científico

costo_supermercado



Método Científico

costo_supermercado






Método Científico y Data Science

Experimento = costo_supermercado



modelo = Tienda

parámetro = distancia

									
1	A	100	220	90	90	110	50	0	660
2	B	110	180	100	90	150	45	50	725
3	C	90	170	90	100	90	40	100	680
4	A	110	242	99	99	121	55	0	726
5	B	121	198	110	99	165	49,5	50	793
6	C	99	187	90	110	90	44	100	720

Features

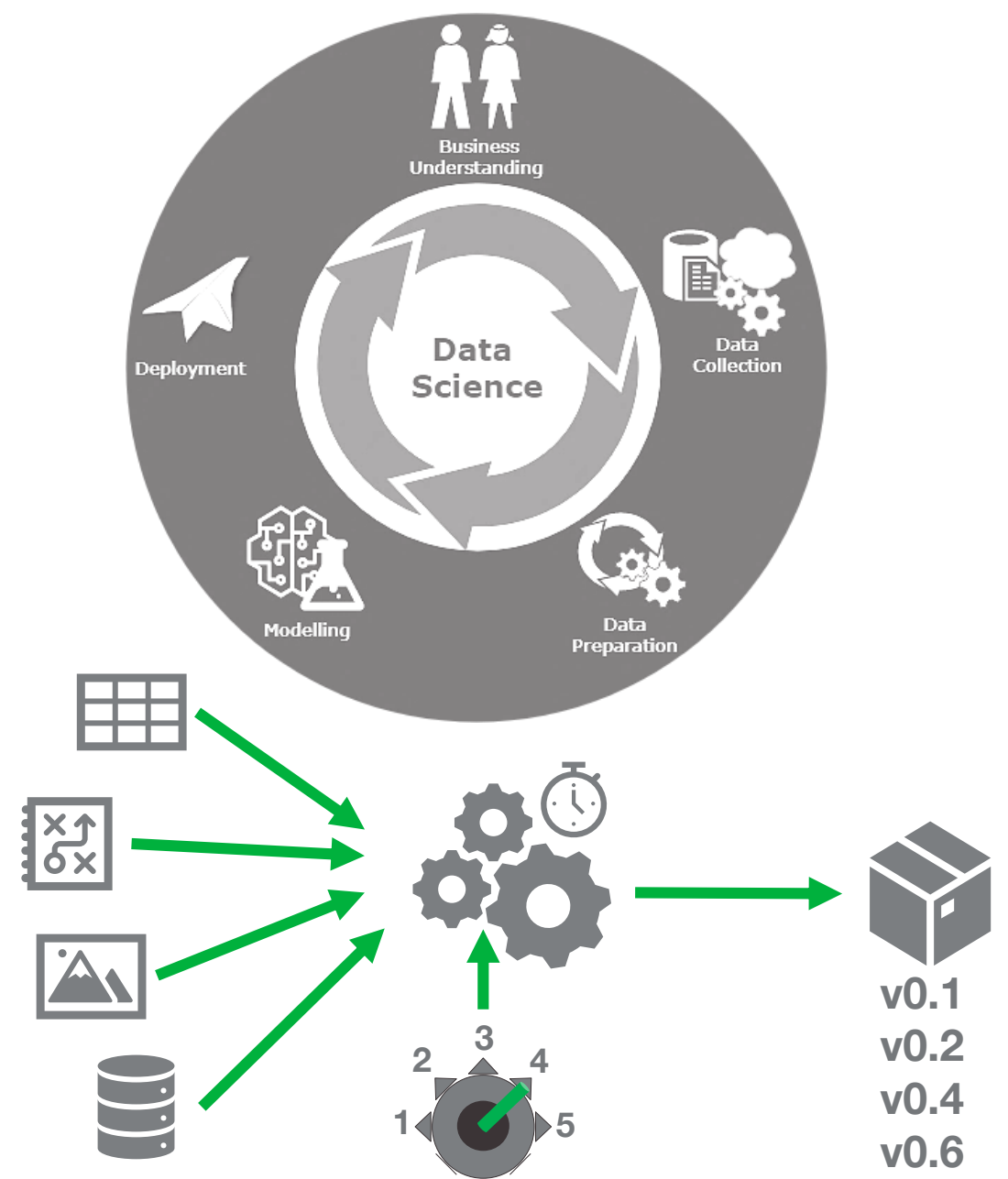
Runs o
ejecuciones

Métrica



Escenario Tecnológico

- El ciclo de vida del desarrollo del ML es muy complejo.
- Surgen muchos problemas que no existen en el ciclo de vida normal del desarrollo de software.
- Hay muchos parámetros de ajuste que cambiar y explorar para obtener un buen modelo.



Escenario Tecnológico



	PARAMETROS				
	DIM (n)	ncons	TOL	EVAL. FUNC.	TIEMPO
43FLOUDAS	3	3	1,00E-03	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-04		MENOS DE 1 MINUTO
			1,00E-05		CASI DOS HORAS
67FLOUDAS	3	1	1,00E-03	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-04	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-05	10000*(n+ncons)	10 MINUTOS
77FLOUDAS	4	2	1,00E-03	10000*(n+ncons)	MENOS DE 10 MINUTOS
			1,00E-04	10000*(n+ncons)	10 MINUTOS
			1,00E-05	10000*(n+ncons)	2 HORAS Y MEDIA
84FLOUDAS	2	1	1,00E-03	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-04		MENOS DE 1 MINUTO
			1,00E-05		MENOS DE 1 MINUTO
94FLOUDAS	2	2	1,00E-03	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-04		MENOS DE 1 MINUTO
			1,00E-05		1 HORA
G8	2	2	1,00E-03	10000*(n+ncons)	MENOS DE 1 MINUTO
			1,00E-04		MENOS DE 1 MINUTO
			1,00E-05		10 MINUTOS
33FLOUDAS	6	6	1,00E-03	10000*(n+ncons)	2 días!

Muy Bueno	77floudas_10000nm_1e-5_190401.txt
Bueno	84floudas_10000nm_1e-3_190401.txt
Muy Bueno	84floudas_10000nm_1e-4_190401.txt
Muy Bueno	84floudas_10000nm_1e-5_190401.txt
Bueno	94floudas_10000nm_1e-3_190401.txt
Muy Bueno	94floudas_10000nm_1e-4_190401.txt
Muy Bueno	94floudas_10000nm_1e-5_190401.txt
	G8_10000nm_1e-3_190401.txt
Muy Bueno	G8_10000nm_1e-4_190401.txt
Muy Bueno	G8_10000nm_1e-5_190401.txt
Bueno	33floudas_10000nm_1e-3_190401.out



Escenario Tecnológico

Los gigantes de la Web han creado plataformas de ciencia de datos.

Ejemplos:

- FBLearner Flow de Facebook.
- TFX de Google.
- Michelangelo de Uber.

Desventajas:

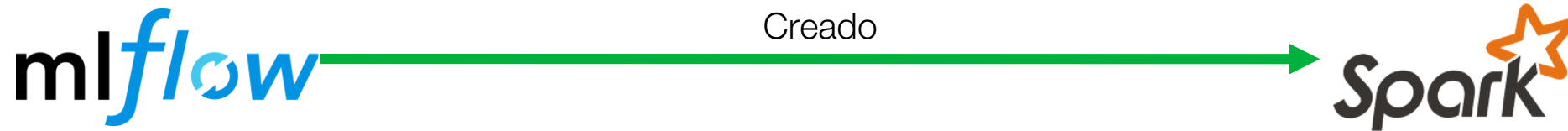
- Se limitan a pocos algoritmos o frameworks.
- Se limitan a funcionar en la infraestructura de la compañía, que limita la capacidad de compartir código.





“platform for the machine learning lifecycle”

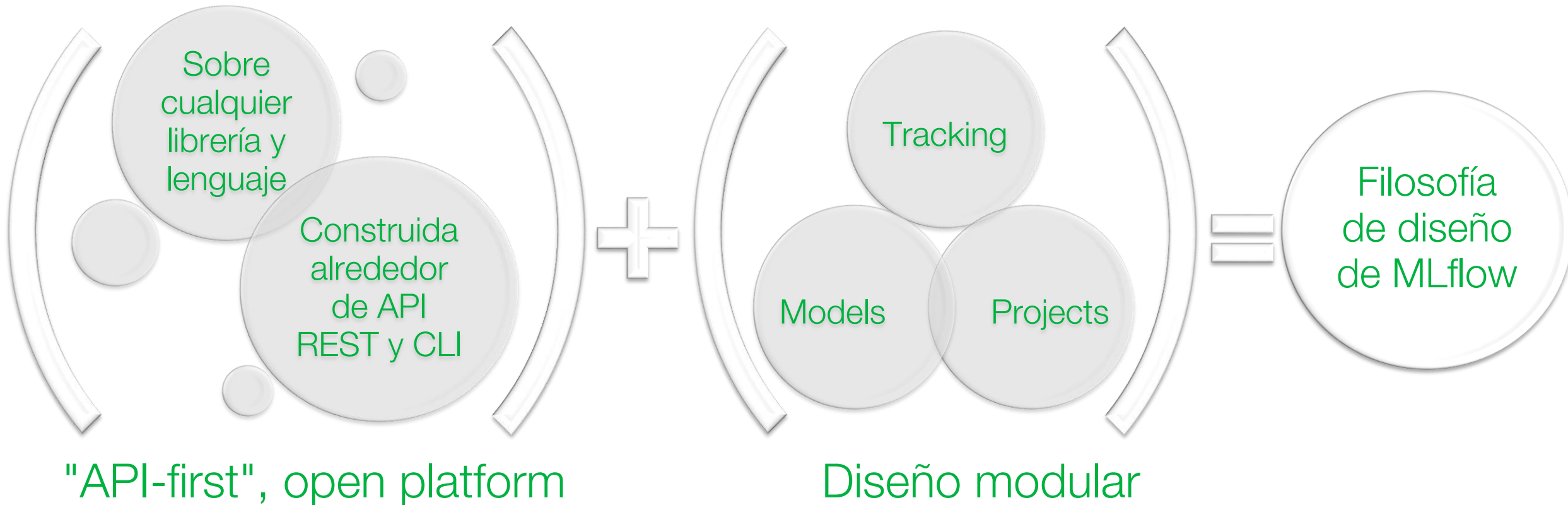
- Presentada por los creadores de Databricks el 5 de junio de 2018.



- Funciona con la mayoría de librería y lenguajes de ML.
- Se ejecuta de la misma manera en cualquier lugar.
- Diseñado para ser útil en organizaciones de 1 o 1000 personas.
- Simple y fácil de usar.



Filosofía de diseño de MLflow



Organización del Proyecto

Experimento



Run 1

- Parámetros
- Métricas
- Modelos



Run 2

- Parámetros
- Métricas
- Modelos



Run 3

- Parámetros
- Métricas
- Modelos



Proceso: Experimento



- El experimento es la unidad principal de organización y de control de acceso para las ejecuciones de MLflow.
- Todas las ejecuciones “runs” de MLflow pertenecen a un experimento.
- Cada experimento le permite visualizar, buscar y comparar ejecuciones, así como descargar artefactos o metadatos para su análisis en otras herramientas.



Proceso: Experimento

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

¿Cómo crear un experimento?

```
mlflow.create_experiment(name='nombre_del_experimento',  
                        artifact_location='La/ubicación/para/almacenar/artefactos/de/ejecución')
```

Si artifact_location es None el servidor elige un valor predeterminado apropiado.

¿Cómo activar el experimento que se quiere usar?

```
mlflow.set_experiment(experiment_name='nombre_del_experimento')
```

En caso de que el experimento no exista se crea uno automáticamente.



Proceso: Run/Ejecución

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

- El run se corresponde con la ejecución del código que hemos considerado como parte del experimento.
- Si se inicia un run sin crear un experimento, mlflow lo crea automáticamente.
- Se puede iniciar y terminar el run de diferentes maneras

Manualmente

```
mlflow.start_run()  
.  
.  
.  
mlflow.end_run()
```

Código del
entrenamiento
del modelo

Usando context manager

```
with mlflow.start_run():  
.  
.  
.
```



Proceso: Log

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

MLflow durante cada ejecución puede registrar la siguiente información:

- *Parámetros*: parámetros del modelo, dataset u otros.
- *Métricas*: métricas de evaluación de modelo. El valor es numérico. Cada métrica puede actualizarse durante el transcurso de la ejecución y MLflow registra y permite visualizar el historial de la métrica.
- *Artefactos*: archivos de salida en cualquier formato. Por ejemplo, puede grabar imágenes, modelos y archivos de datos.
- *Origen*, *Versión*, *Hora de inicio & fin*, *Etiquetas*.



Proceso: Log

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

Para guardar los parámetros usados:

```
mlflow.log_param(key='nombre_del_parametro', value='valor del parametro')
```

Para guardar las métricas obtenidas:

```
mlflow.log_metric(key='nombre_de_la_metrica', value='valor_de_la_metrica')
```

Para guardar los artifact (imágenes, modelos, etc.):

```
mlflow.log_artifact(local_dir='direccion/donde/se/va/a/guardar/el/artifact',  
                    artifact_path=None)
```



Proceso: Log Model

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

Mlflow también tiene una parte dedicada al registro de modelos que se pueden guardar entrenados y después accederá ellos directamente para predecir. Tiene funciones dedicadas a librerías más usadas.

- ✓ H2O (h2o)
- ✓ Keras (keras)
- ✓ MLeap (mleap)
- ✓ PyTorch (pytorch)
- ✓ Scikit-learn (sklearn)
- ✓ Spark MLlib (spark)
- ✓ TensorFlow (tensorflow)
- ✓ ONNX (onnx)
- ✓ MXNet Gluon (gluon)
- ✓ XGBoost (xgboost)
- ✓ LightGBM (lightgbm)
- ✓ R Function(crate)



Proceso: Log Model

Experimento



Run 1

- Parámetros
- Métricas
- Modelos



Run 2

- Parámetros
- Métricas
- Modelos



Run 3

- Parámetros
- Métricas
- Modelos

Por default guarda tres archivos:

- python_model.pkl
- conda.yaml
- MLmodel

Dirección donde se va a guardar (por default el origen es el directorio del artifact)

```
mlflow.libreria_que_se_uso.log_model(modelo, artifact_path, conda_env=None)
```

Librería usada
(sklearn, spark, etc)

Modelo ya entrenado
que se desea guardar

Puede ser una representación de diccionario de un entorno Conda o la ruta a un archivo yaml del entorno Conda (Si no hay ninguno, el entorno predeterminado `get_default_conda_env()` se agrega al modelo)



Proceso: Ejemplo

Creación del
bloque para
comenzar el run

Creación del
experimento
de nombre iris

```
mlflow.set_experiment('iris')
with mlflow.start_run():
    from sklearn import datasets
    iris_X, iris_y = datasets.load_iris(return_X_y=True)
    np.unique(iris_y)
    np.random.seed(0)
    indices = np.random.permutation(len(iris_X))
    iris_X_train = iris_X[indices[:-10]]
    iris_y_train = iris_y[indices[:-10]]
    iris_X_test = iris_X[indices[-10:]]
    iris_y_test = iris_y[indices[-10:]]

    # Create and fit a nearest-neighbor classifier
    from sklearn.neighbors import KNeighborsClassifier
    n = 10
    mlflow.log_param('vecinos', n)
    knn = KNeighborsClassifier(n_neighbors = n)
    knn.fit(iris_X_train, iris_y_train)
    knn.predict(iris_X_test)
    score = knn.score(iris_X_test, iris_y_test)
    mlflow.log_metric('score_iris', score)
    mlflow.sklearn.log_model(knn, 'knn')
```

Guardado de la
métrica score
con el nombre
de score_iris

Guardado del
parámetro n con el
nombre de vecinos

Guardado del
modelo knn en el
directorio knn



Proceso: Ver Resultados

Podemos acceder a toda la información que hemos logeado durante una ejecución con *search_runs()*:

```
mlflow.search_runs()
```

	run_id	experiment_id	status	artifact_uri	start_time	end_time	metrics.bic	metrics.dickey- fuller-test	metrics.aic	mel ob
0	1fc4c13276114437ba6e2b76f048733a	1	FINISHED	file:///user-home/1027/DSX_Projects/Analisis%20-%20Descriptivo/jupyter/mlruns/1/e4d98a63c5844f6f922128bb9271a23f/artifacts	2020-01-20 14:52:30.058000+00:00	2020-01-20 14:52:30.637000+00:00	104.994338	1.180787e-10	103.783998	

Podemos obtener la dirección donde se guardan los artifact de cada experimento de la siguiente manera:

```
mlflow.search_runs().artifact_uri[0]
```

```
'file:///user-home/1027/DSX_Projects/Analisis%20-%20Descriptivo/jupyter/mlruns/1/e4d98a63c5844f6f922128bb9271a23f/artifacts'
```



Proceso: Cargar el modelo

Obtener ruta del modelo de un determinado run:

```
model_uri = mlflow.search_runs(\
    [mlflow.search_runs()['run_id'] == 'c7e11be70d55448fa81b34f55d1ebb82']\
    .artifact_uri.item()
model_uri
```

```
'file:///C:/Users/.../.../...e/...p
.../WiDS/mlruns/1/c7e11be70d55448fa81b34f55d1ebb82/artifacts'
```

Cargar el modelo con la función load_model:

```
model = mlflow.sklearn.load_model(model_uri+'/knn')
```

```
type(model)
```

```
sklearn.neighbors._classification.KNeighborsClassifier
```

← Notar que el modelo es de sklearn



Proceso: Pyfunc

¿Cómo guardamos un modelo que no pertenezcan a alguna de las librerías anteriores en python?



La respuesta es
pyfunc



Proceso: Pyfunc

- Define un formato de sistema de archivos genérico para los modelos Python.
- Proporciona utilidades para guardar y cargar modelos desde y hacia este formato.
- Es autónomo en el sentido de que incluye toda la información necesaria para cargar y usar un modelo.
- Las dependencias se almacenan directamente con el modelo o se hace referencia a través del entorno Conda.



Proceso: Pyfunc

¿Cómo se guarda un modelo utilizando pyfunc?

Necesitamos definir una clase

```
class SARIMA(mlflow.pyfunc.PythonModel):  
  
    def __init__(self, model):  
        self.model = model  
        super(SARIMA, self).__init__()  
  
    def load_context(self, context):  
        import statsmodels.api as sm  
        return  
  
    def predict(self, context, model_input):  
        future = self.model.get_forecast(model_input)  
        future = future.predicted_mean  
        return future
```

Función `__init__` necesaria para la creación de la clase

Dependencias del modelo usado

Método necesario para que, una vez guardado, el modelo se pueda usar para predecir



Manos a la Obra



Conclusión

Experimento



Run 1
• Parámetros
• Métricas
• Modelos



Run 2
• Parámetros
• Métricas
• Modelos



Run 3
• Parámetros
• Métricas
• Modelos

create_experiment()
set_experiment()

Setear
experimento

Comenzar
un run

start_run()
end_run()

log_param()
log_metric()
log_artifact()

Registrar
información
relevante

library.log_model()

Registrar
modelo

list_experiments()
search_runs()
library.load_model()

Cargar
modelo



Ahora sabes porque MLFlow puede ser TU mejor amiga!!!

Repo:

<https://github.com/PiConsulting>

En redes búscame como Melina Solovey



¡Muchas Gracias!

