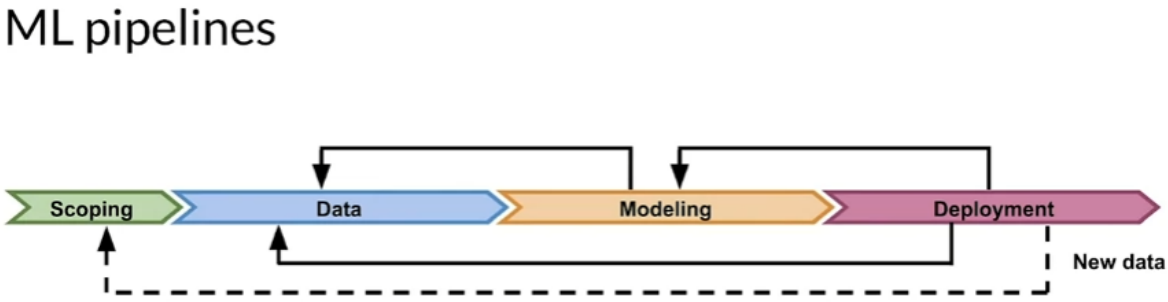
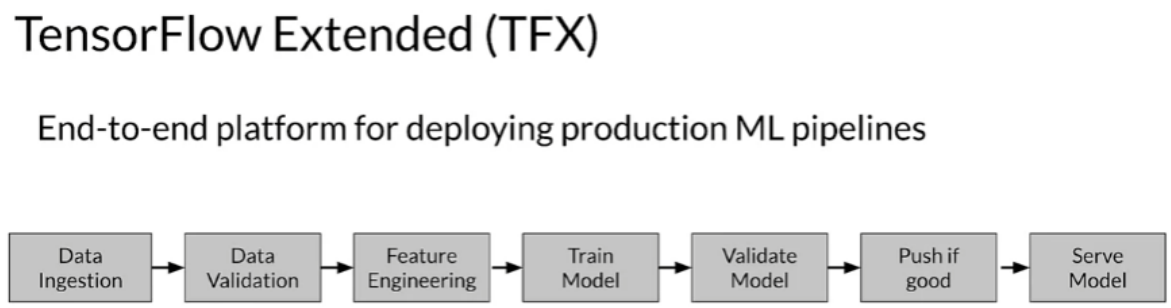
1. **<u>Semana 1</u>**

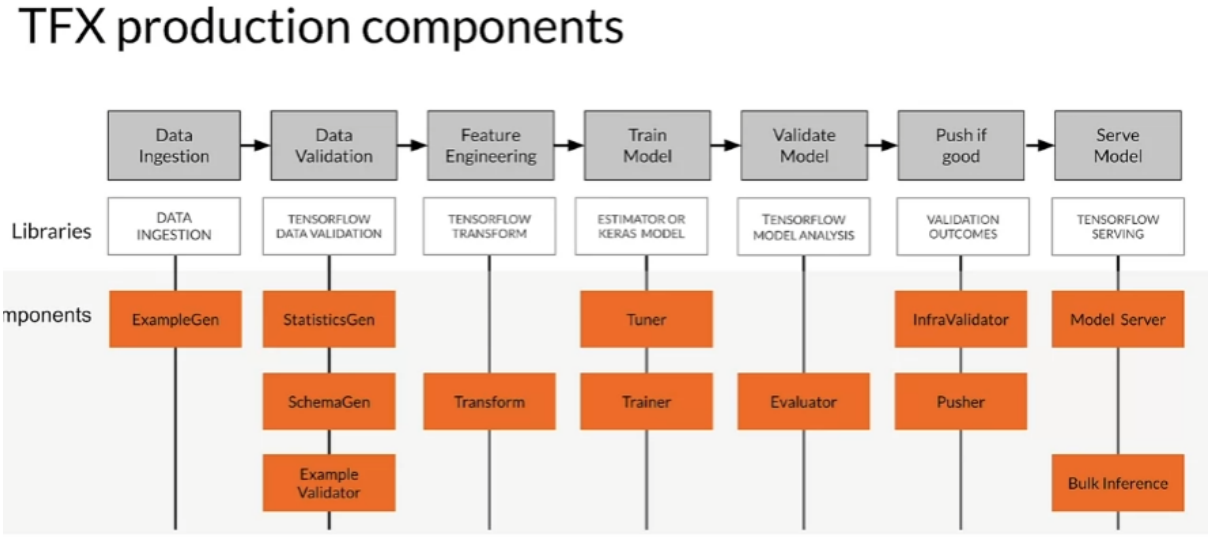
- Há uma diferença entre os dados para pesquisa e para produção, no caso da produção, estão sempre mudando, o importante é ter uma boa inferência e uma boa interpretabilidade, enquanto num ambiente acadêmico o importante é uma acurácia elevada e os dados são estáticos geralmente.
- Fatores como modularidade, extensibilidade, teste de módulo e de código inteiro se tornam bem mais importantes em um modelo para produção
- ML Pipeline: infraestrutura seguida para otimizar, monitorar e manter modelos para treinamento e deploy



- As pipelines geralmente são feitas em formas de DAGs (directed acyclic graphic) que é um gráfico sem ciclos e define a sequência de tarefas a serem perfomadas.
- Tensorflow Extended TEX é um pipeline end to end para deploy de produtos de ML



-



-

- Existem vários tipos de problemas quando vamos classificar os dados, os dados podem ficar obsoletos com o tempo (slow problems) ou terem um sensor ruim, software sofreu atualização, etc (fast problems)
- Além disso, deve-se considerar sempre o bias e custo quando vai se classificar, pensar em quem classificou, se tem uma visão ocidental da sociedade, por exemplo, e a partir disso tentar medir a qualidade dos dados
- Além disso, é sempre bom lembrar coisas como, se algo é classificado como moda em um ano, no outro pode não ser, isso faz com que o sistema antes implementado necessite de alterações.
- Para se detectar anomalias num dataset pode se user o TFDV para capturar estatisticas e erros de um dataset: https://www.coursera.org/learn/machine-learning-data-lifecycle-in-production/ungradedLab/KUstm/tfdv-exercise/lab?path=%2Fnotebooks%2FC2_W1_Lab_1_TFDV_Exercise.ipynb

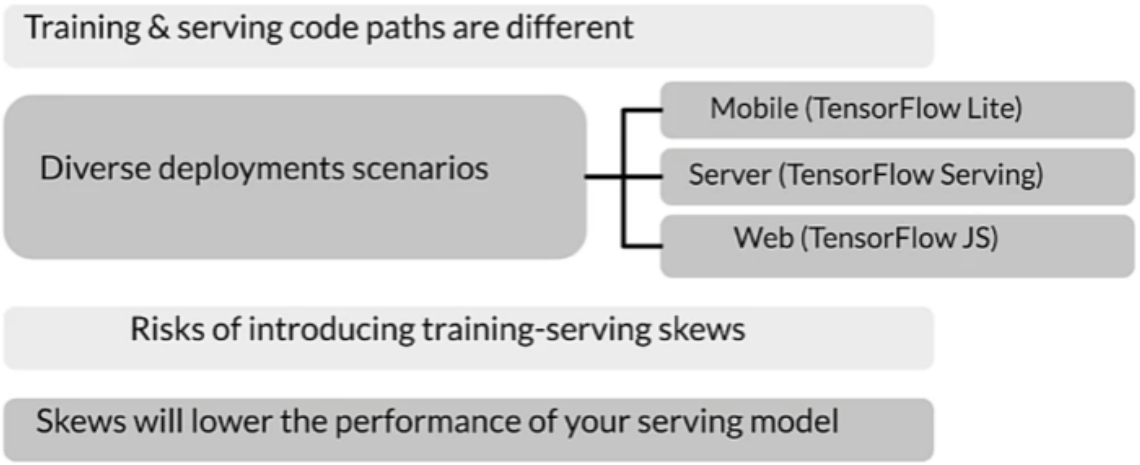
2. Semana 2 - Pré-processamento

- Feature engineering: tenta aumentar o aprendizado do modelo enquanto se possível reduz os recursos computacionais necessários. Deve-se espremer o máximo dos dados, aplicando sempre que novos dados foram postos no dataset e ajustando os parâmetros quando isso acontece
- Existem algumas operações principais quando se lida com features, como data cleansing, feature tuning, transformation, feature extraction, feature construction…
- Scaling: escalarizar os números, ajuda a NN a converter mais rápido e encontrar os pesos corretos, normalization é bom se a distribuição não é gaussiana, standarization(z-score) usa o desvio padrão e concentra os

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \qquad \text{(z-score)}$$

$$X_{\text{std}} \sim \mathcal{N}(0, \sigma)$$

- Bucketing/binning: Por exemplo usar um one hot enconder para separar datas > 2000 por exemplo
- Feature crosses: Combinar múltiplos features em uma nova, pode ser útil para evitar redundâncias e coisas do tipo, desde que dados não sejam omitidos/ignorados
- Feature engineering em escala industrial traz outros problemas, como inconsistências, otimizações, granularidade do pré-processamento, podem ser terabytes de dados, os frameworks para esses datasets são diferentes, deve se haver uma consistência no pré processamento sempre que dados novos entrarem

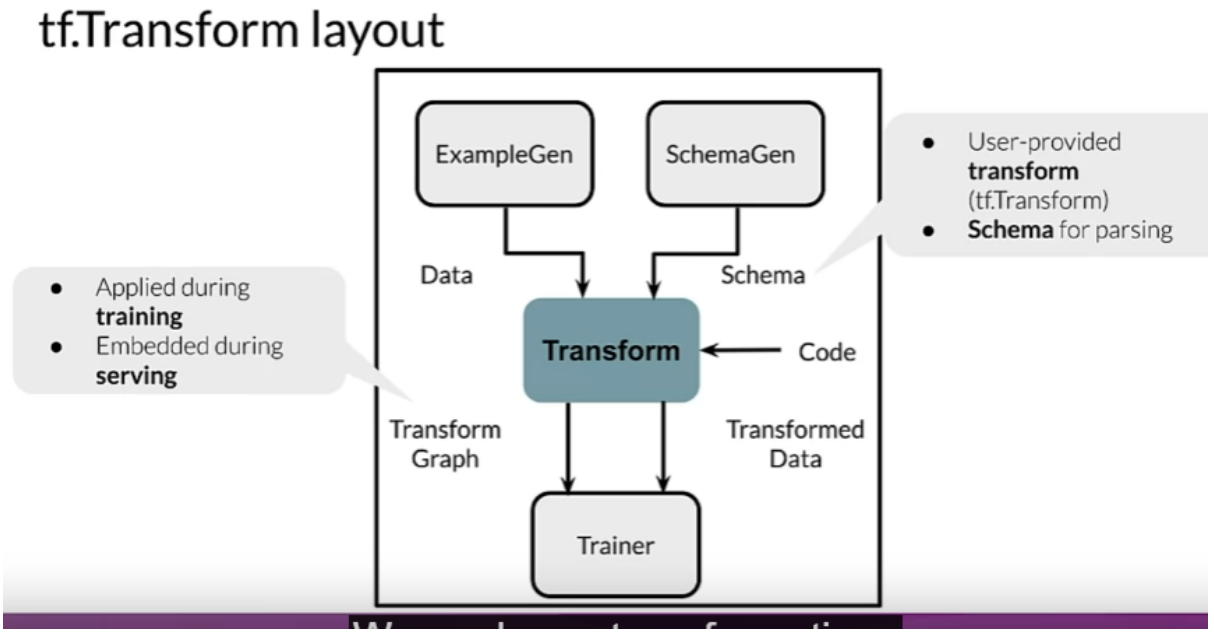


-
- Deve-se prestar atenção na diferença de coisas que se faz em um dataset inteiro e em um instance-level (numa instância nova), coisas como clipping, expandir features, min max, etc.Não é a mesma coisa e computacionalmente não é eficaz ficar sempre pegando o dataset inteiro para trabalhar quando se coloca uma instância nova

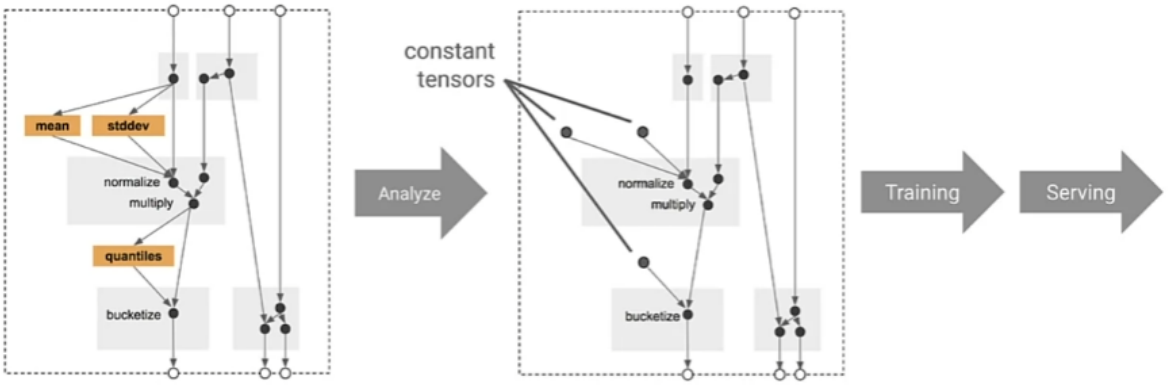
Transformations	
Instance-level	**Full-pass**
Clipping	Minimax
Multiplying	Standard scaling
Expanding features	Bucketizing
etc.	etc.

-
- Pode-se fazer essas transformações por batch e evitar de entrar no dataset inteiro, porém deve-se tomar cuidado no tamanho e no bias de cada batch, além do tamanho, claro.
- **TensorFlow transform** framework utilizado para pré processar os dados, recebe como entrada os dados em forma de examplegen e um schemagen na forma de schema e então transforma a partir disso. Retorna um transform graph que representa todas as transformaçõe sque fizemos nos dados junto dos dados

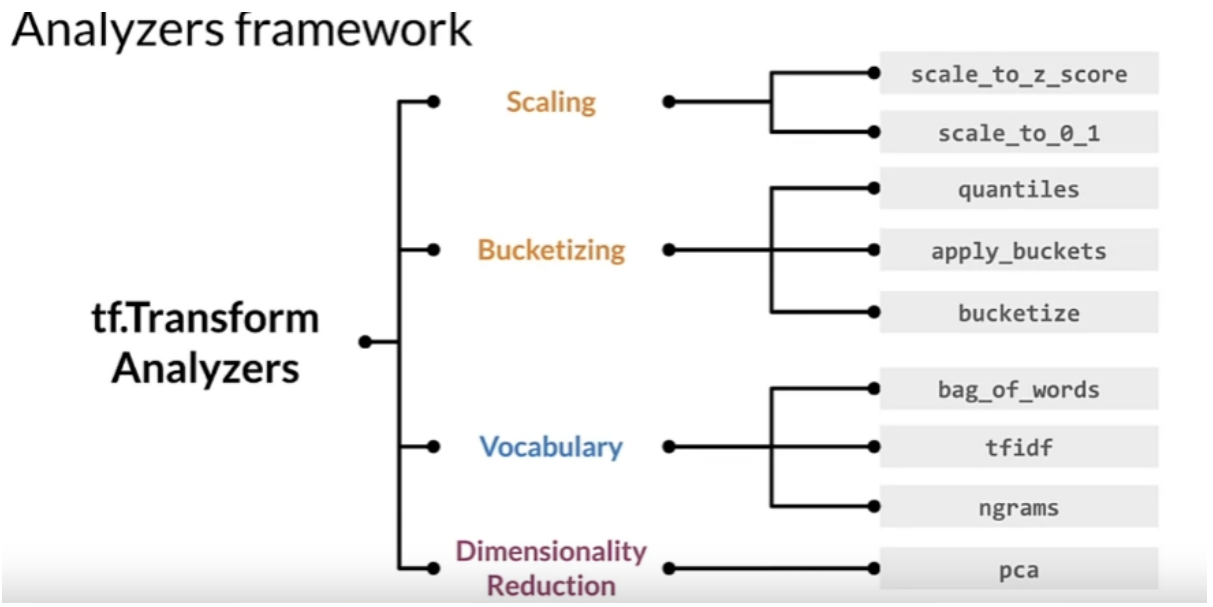
transformados

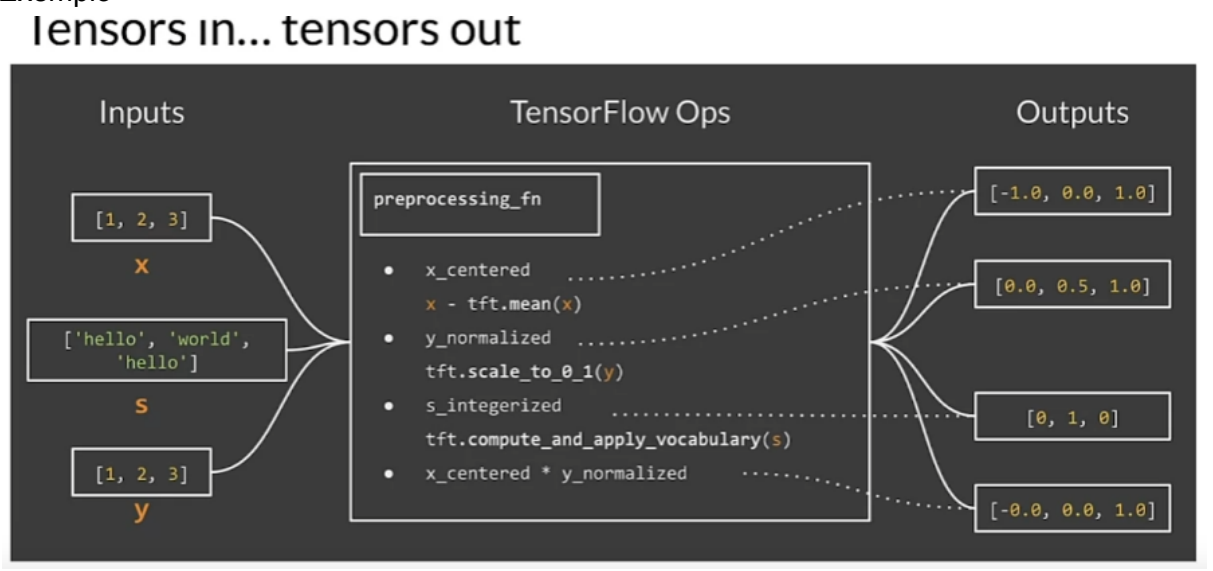


- Existem operações no transform como o min que computa o minimo de umt ensor uma vez apenas durante o treinamento, faz isso de uma maneira pros grafos, de modo a alterar esses grafos sempre que necess[arios, não necessitando passar pelo dataset inteiro. tf.graph segura todas constantes e transformaçôes, funciona inline, não precisa do pre processamento

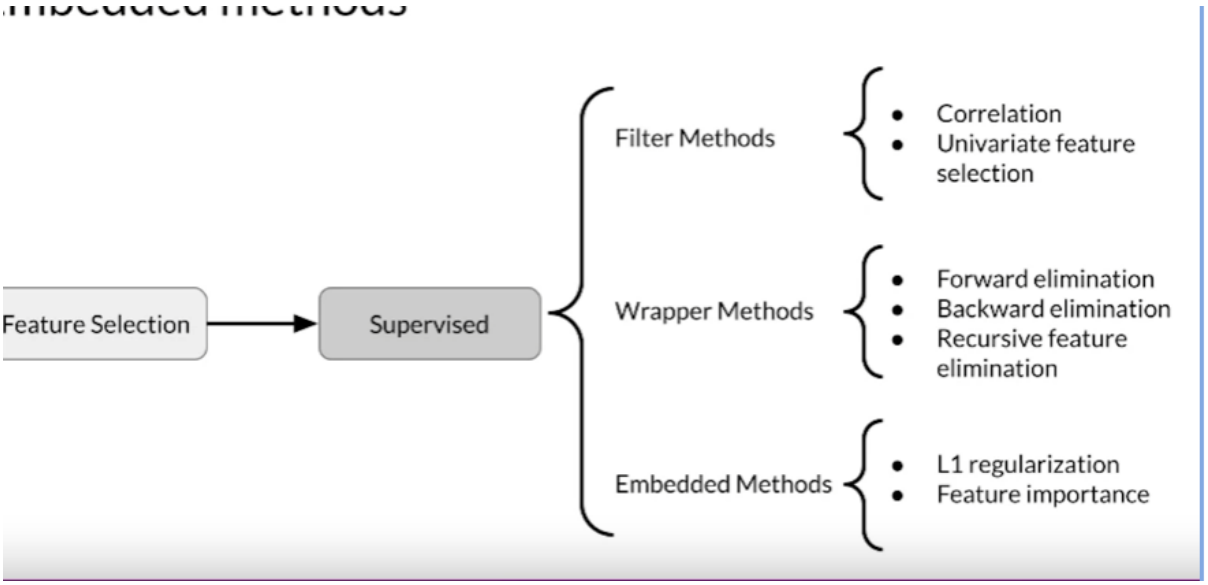


-

- 

- Exemplo
- 

- Feature space: Espaço N dimensional no qual as N features estão definidas, não inclui a target label
- Recursive Feature Elimination(RFE): Método para diminuir o número de features, seleciona um modelo e avalie as features com menor importância, selecione a quantidade de features que deseja e descarta as menos importantes
- Há outras formas de fazer isso, por exemplo com correlação.



- É possível usar o SKLearn com randomforest para ver a importância das features

```
def select_features_from_model(model):


  model = SelectFromModel(model, prefit=True, threshold=0.012)

  feature_idx = model.get_support()
  feature_names = df.drop("diagnosis_int",1 ).columns[feature_idx]
  return feature_names
```

Semana 3:

- **Data journey**
- Algumas definições: artefatos - é tudo que é prodzudido, são criados quando os componentes do pipeline são executados, inclui todos os dados e objetos produzidos por componentes do pipeline
- Data provenance é a sequência de objetos criados conforme se movemos pelo pipeline, é associado com o código e o componente que criamos, nos ajuda a debugar, voltar no código e entender o que cada etapa fez
- **Metadata:** ajuda a trackear o progesso, sempre que uma atualização é feita é armazenado essa informação nos metadados, há uma biblioteca para isso (ml-metadata), que traqueia os metados entre os componentes do pipeline. Armazena dados como os inputs e outputs gerados por um componente e sua execução, normalmente essas informações são armazenados em um componente externo como um banco de dados,a biblioteca fornece opções como um fake database(na memória mesmo), sqlite, mysql, etc.
- **Schema:** É um resumo das features em um dataset, incluem nome, tipo de variável, a valência, aplicabilidade da feature, min e max de cada uma, etc. O schema deve ser mantido, prestando atenção na evolução dos dados, checando anomalias e etc

no dataset. exemplo de uso do schema para detecção de anomalias

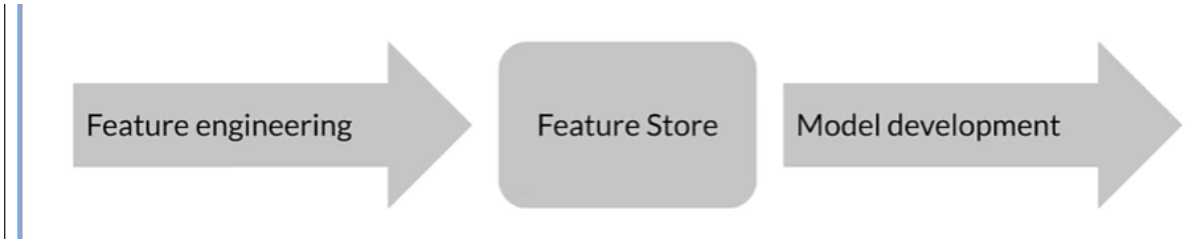
Inspect anomalies in serving dataset

```
stats_options = tfdv.StatsOptions(schema=schema,
                                  infer_type_from_schema=True)

eval_stats = tfdv.generate_statistics_from_csv(
    data_location=SERVING_DATASET,
    stats_options=stats_options
)

serving_anomalies = tfdv.validate_statistics(eval_stats, schema)
tfdv.display_anomalies(serving_anomalies)
```

- **Feature Store:** Repositório central para armazenamento de features bem documentadas e de acesso controlado, permite que times compartilhem e usem features bem curadas, permite facilidade no descobrimento e uso de features. Pode ser visto como a interface entre feature engineering e model development. Evitam duplicação, fazem um controle de acesso também, permitem uma abordagem offline dos dados, permitindo monitoramento offline por exemplo, em casos online deve-se ter tudo muito rápido, com pouca latência , pode-se usar de batches ou features pré carregadas, tem que ser simples e eficiente



- **Data Warehouse**: agregas dados de diversas fontes, não é em tempo real, faz um processamento dos dados,necessita de um esquema, é orientado pelos sujeitos que fazem essas inserções de dados, oferece abilidade de analisar dados, poermite consist~enciaq e qualidade dos dados, aumenta perfomance por permitir consultas rápidas, são muito grandes, maiores que

databases

Comparison with databases

Data warehouse	Database
Online analytical processing (OLAP)	Online transactional processing (OLTP)
Data is refreshed from source systems	Data is available real-time
Stores historical and current data	Stores only current data
Data size can scale to >= terabytes	Data size can scale to gigabytes
Queries are complex, used for analysis	Queries are simple, used for transactions
Queries are long running jobs	Queries executed almost in real-time
Tables need not be normalized	Tables normalized for efficiency

- **Data lake:** é um sistema ou repositório de data que armazena na forma natural e crua os dados, geralmente em forma de arquivos. Como um data warehouse, agrega dados de várias fontes de dados. Pode incluir arquivos estruturados ou não estruturados, não envolvem processamento e schemas, não há destino ainda pros dados geralmente

Comparison with data warehouse

	Data warehouses	Data lakes
Data Structure	Processed	Raw
Purpose of data	Currently in use	Not yet determined
Users	Business professionals	Data scientists
Accessibility	More complicated and costly to make changes	Highly accessible and quick to update

- **Active learning:** Familia de algoritmos para inteligentemente samplear dados Seleciona pontos para serem classificados que sejam mais informativos para o treinamento de modelo

Active learning sampling techniques

Margin sampling: Label points the current model is least confident in.

Cluster-based sampling: sample from well-formed clusters to "cover" the entire space.

Query-by-committee: train an ensemble of models and sample points that generate disagreement.

Region-based sampling: Runs several active learning algorithms in different partitions of the space.

-

- Semi-supervised learning:
  - Applies the best of supervised and unsupervised approaches
  - Using a small amount of labeled data boosts model accuracy
- Active learning:
  - Selects the most important examples to label
  - Improves predictive accuracy
- Weak supervision:
  - Uses heuristics to apply noisy labels to unlabeled examples
  - Snorkel is handy framework for weak supervision

-
-