Kaggle Learning Course

이재욱, 이한나, 이나경, 이태경

Project Proposal

Topic | SQL / Deep Learning / Data Cleaning / AI / Geospatial Analysis

Description | Kaggle 에 있는 수업들을 수강하여 데이터 사이언스 안에 다양한 분야 (SQL, Deep Learning, AI Ethics, Geospatial Analysis, Game AI) 를 배울 이후 Kaggle 수업이 끝나면 배웠던 분야 중 제일 관심이 깊었던 분야에 관한 데이터 프로젝트를 진행할 예정

Expected Duration | 5 - 6 Week

Team Member | 이재욱, 이한나, 이나경, 이태경

Timeline

Week 5

- Project Proposal
- Lesson Overview

Week 6

- Intro to SQL
- Advanced SQL

Week 7

- Deep Learning
- Data Cleaning

Week 8

- Time Series
- Geospatial Analysis

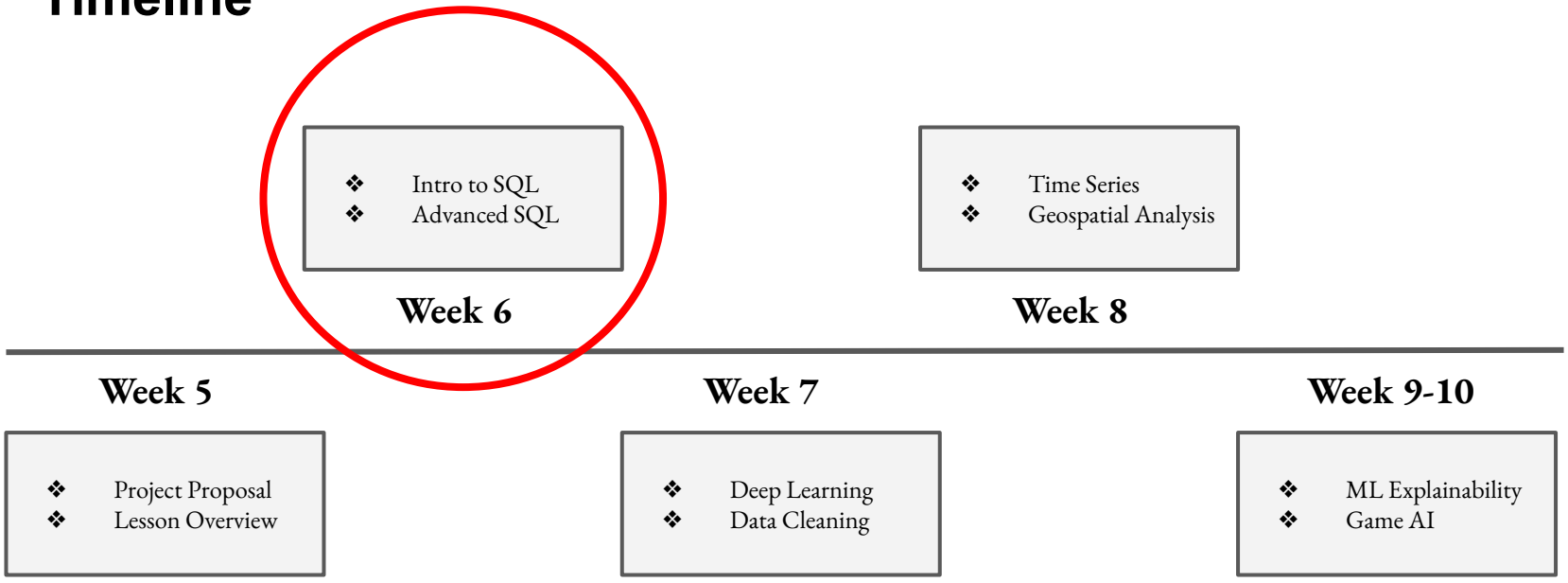
Week 9-10

- ML Explainability
- Game AI

SQL and BigQuery

이재욱, 이한나, 이나경, 이태경

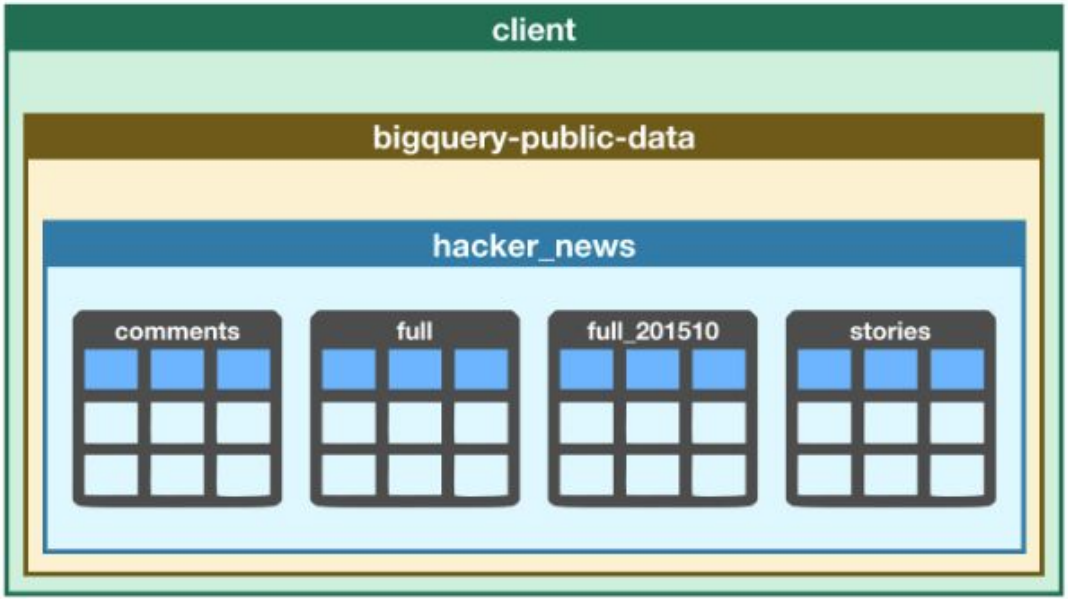
Timeline


Intro to SQL
Advanced SQL
Week 6
Time Series
Geospatial Analysis
Week 8
Week 5
Project Proposal
Lesson Overview
Week 7
Deep Learning
Data Cleaning
Week 9-10
ML Explainability
Game AI

SQL vs Python

SQL	Python
● Programming language to manage and retrieve data from databases ● Focus is to ‘create’ and ‘maintain’ the datasets ● Access and extract data from database ● Not designed for high level data ● Industry Standard ● Querying data is faster than Python (Simple)	● Analyze and manipulate data ● Specifically designed to create applications

SQL Structure



Client objects hold projects and a connection to the BigQuery service.

bigquery-public-data is a **project**. A project is a collection of datasets.

hacker_news is a **dataset**. A dataset is a collection of tables.

comments, *full*, *full_201510*, and *stories* are **tables**.

Select, From, Where

: get data from specifically based on conditions

```
query = """
        SELECT Name
        FROM `bigquery-public-data.pet_records.pets`
        WHERE Animal = 'Cat'
        """
```

ID	Name	Animal
1	Dr. Harris Bonkers	Rabbit
2	Moon	Dog
3	Ripley	Cat
4	Tom	Cat

Group By, Having & Count

: to group data and count things within those groups

ID	Name	Animal
1	Dr. Harris Bonkers	Rabbit
2	Moon	Dog
3	Ripley	Cat
4	Tom	Cat

```
query = """
        SELECT COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        """
```

f0_
4

```
query = """
        SELECT Animal, COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        GROUP BY Animal
        """
```

Animal	f0_
Rabbit	1
Dog	1
Cat	2

Group By, Having & Count

: to group data and count things within those groups

ID	Name	Animal
1	Dr. Harris Bonkers	Rabbit
2	Moon	Dog
3	Ripley	Cat
4	Tom	Cat

```
query = """
        SELECT Animal, COUNT(ID)
        FROM `bigquery-public-data.pet_records.pets`
        GROUP BY Animal
        HAVING COUNT(ID) > 1
        """
```

Animal	f0_
Cat	2

Order By

: sorts the results returned by the rest of your query

ID	Name	Animal
1	Dr. Harris Bonkers	Rabbit
4	Tom	Cat
2	Moon	Dog
3	Ripley	Cat

```
query = """
        SELECT ID, Name, Animal
        FROM `bigquery-public-data.pet_records.pets`
        ORDER BY ID
        """
```

ID	Name	Animal
1	Dr. Harris Bonkers	Rabbit
2	Moon	Dog
3	Ripley	Cat
4	Tom	Cat

JOIN and UNIONS

`owners` table

ID	Name	Age	Pet_ID
1	Aubrey Little	20	1
2	Chett Crawfish	45	3
3	Jules Spinner	10	4
4	Magnus Burnsides	9	2
5	Veronica Dunn	8	NULL

`pets` table

ID	Name	Age	Animal
1	Dr. Harris Bonkers	1	Rabbit
2	Moon	9	Dog
3	Ripley	7	Cat
4	Tom	2	Cat
5	Maisie	10	Dog

Dr. Harris Bonkers is owned by **Aubrey Little**.

Moon is owned by **Magnus Burnsides**.

Ripley is owned by **Chett Crawfish**.

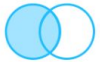
Tom is owned by **Jules Spinner**.

Veronica Dunn does not have a pet.
Maisie does not have an owner.

INNER JOIN

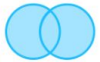
Owner_Name	Pet_Name
Aubrey Little	Dr. Harris Bonkers
Magnus Burnsides	Moon
Chett Crawfish	Ripley
Jules Spinner	Tom

LEFT JOIN



Owner_Name	Pet_Name
Aubrey Little	Dr. Harris Bonkers
Magnus Burnsides	Moon
Chett Crawfish	Ripley
Jules Spinner	Tom
Veronica Dunn	NULL

FULL JOIN



Owner_Name	Pet_Name
Aubrey Little	Dr. Harris Bonkers
Magnus Burnsides	Moon
Chett Crawfish	Ripley
Jules Spinner	Tom
Veronica Dunn	NULL
NULL	Maisie

UNION

```
query = """
        SELECT Age FROM `bigquery-public-data.pet_records.pets`
        UNION ALL
        SELECT Age FROM `bigquery-public-data.pet_records.owners`
        """
```

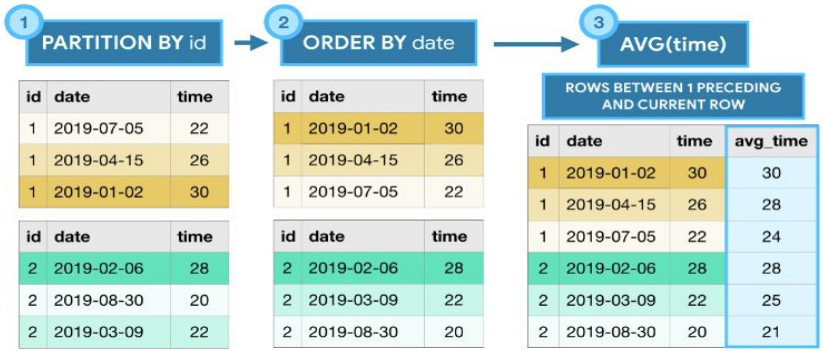
Age
20
45
10
9
8
1
9
7
2
10

Analytic Functions

id	date	time
1	2019-07-05	22
1	2019-04-15	26
2	2019-02-06	28
1	2019-01-02	30
2	2019-08-30	20
2	2019-03-09	22

```
query = """
        SELECT *,
            AVG(time) OVER(
                                PARTITION BY id
                                ORDER BY date
                                ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
                                ) as avg_time
        FROM `bigquery-public-data.runners.train_time`
        """
```

id	date	time
1	2019-07-05	22
1	2019-04-15	26
2	2019-02-06	28
1	2019-01-02	30
2	2019-08-30	20
2	2019-03-09	22



id	date	time
1	2019-07-05	22
1	2019-04-15	26
1	2019-01-02	30

id	date	time
2	2019-02-06	28
2	2019-08-30	20
2	2019-03-09	22

id	date	time
1	2019-01-02	30
1	2019-04-15	26
1	2019-07-05	22

id	date	time
2	2019-02-06	28
2	2019-03-09	22
2	2019-08-30	20

id	date	time	avg_time
1	2019-01-02	30	30
1	2019-04-15	26	28
1	2019-07-05	22	24
2	2019-02-06	28	28
2	2019-03-09	22	25
2	2019-08-30	20	21

Analytic Functions

Aggregate

- MIN / MAX
- AVG / SUM
- COUNT

Navigation

- FIRST_VALUE / LAST
- LEAD / LAG

Numbering

- ROW_NUMBER
- RANK

Nested and Repeated Data

Nested Fields

- Nested columns have type **STRUCT**

Repeated Fields

- Table permits more than **one value** for each row

Nested & Repeated

- Table Combines nested and repeated fields denormalized many relationship without joins

`pets` table

ID	Name	Age	Animal
1	Moon	9	Dog
2	Ripley	7	Cat
3	Napoleon	1	Fish

`toys` table

ID	Name	Type	Pet_ID
1	McFly	Frisbee	1
2	Fluffy	Feather	2
3	Eddy	Castle	3

vs.

`pets_and_toys` table

ID	Name	Age	Animal	Toy
1	Moon	9	Dog	{**Name**: McFly, **Type**: Frisbee}
2	Ripley	7	Cat	{**Name**: Fluffy, **Type**: Feather}
3	Napoleon	1	Fish	{**Name**: Eddy, **Type**: Castle}

`pets` table

ID	Name	Age	Animal
1	Moon	9	Dog
2	Ripley	7	Cat
3	Napoleon	1	Fish

`toys_type` table

ID	Type	Pet_ID
1	Frisbee	1
2	Bone	1
3	Rope	1
4	Feather	2
5	Ball	2
6	Castle	3

vs.

`pets_and_toys_type` table

ID	Name	Age	Animal	Toys
1	Moon	9	Dog	[Frisbee, Bone, Rope]
2	Ripley	7	Cat	[Feather, Ball]
3	Napoleon	1	Fish	[Castle]

`pets` table

ID	Name	Age	Animal
1	Moon	9	Dog
2	Ripley	7	Cat
3	Napoleon	1	Fish

`more_toys` table

ID	Name	Type	Pet_ID
1	McFly	Frisbee	1
2	Scully	Bone	1
3	Pusheen	Rope	1
4	Fluffy	Feather	2
5	Robert	Ball	2
6	Eddy	Castle	3

vs.

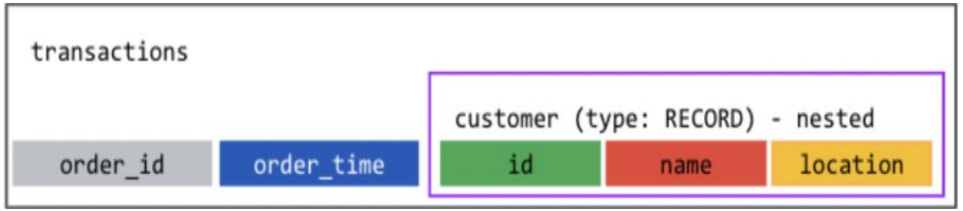
`more_pets_and_toys` table

ID	Name	Age	Animal	Toys
1	Moon	9	Dog	[{**Name**: McFly, **Type**: Frisbee}, {**Name**: Scully, **Type**: Bone}, {**Name**: Pusheen, **Type**: Rope}]
2	Ripley	7	Cat	[{**Name**: Fluffy, **Type**: Feather}, {**Name**: Robert, **Type**: Ball}]
3	Napoleon	1	Fish	[{**Name**: Eddy, **Type**: Castle}]

Nested and Repeated Data

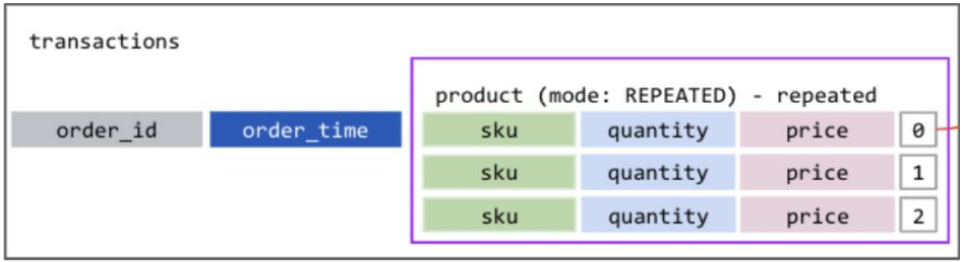
Nested Fields

- Table is collapsed into a single column



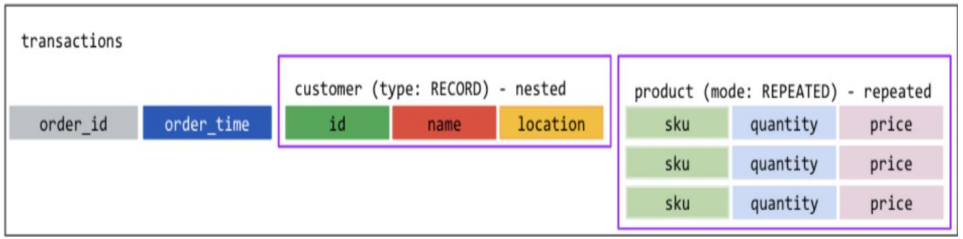
Repeated Fields

- Permits more than one value for each row



Nested & Repeated

- Combining nested and repeated fields denormalized many relationship without joins



Writing Efficient Queries

- Database systems have a **query optimizer** that attempts to interpret/execute query in the most effective way
    - *show_amount_of_data_scanned()* shows the amount of data query used
    - *show_time_to_run() prints* how long it takes for the query to execute
- Strategies
    - Only select the columns you want
    - SELECT * FROM

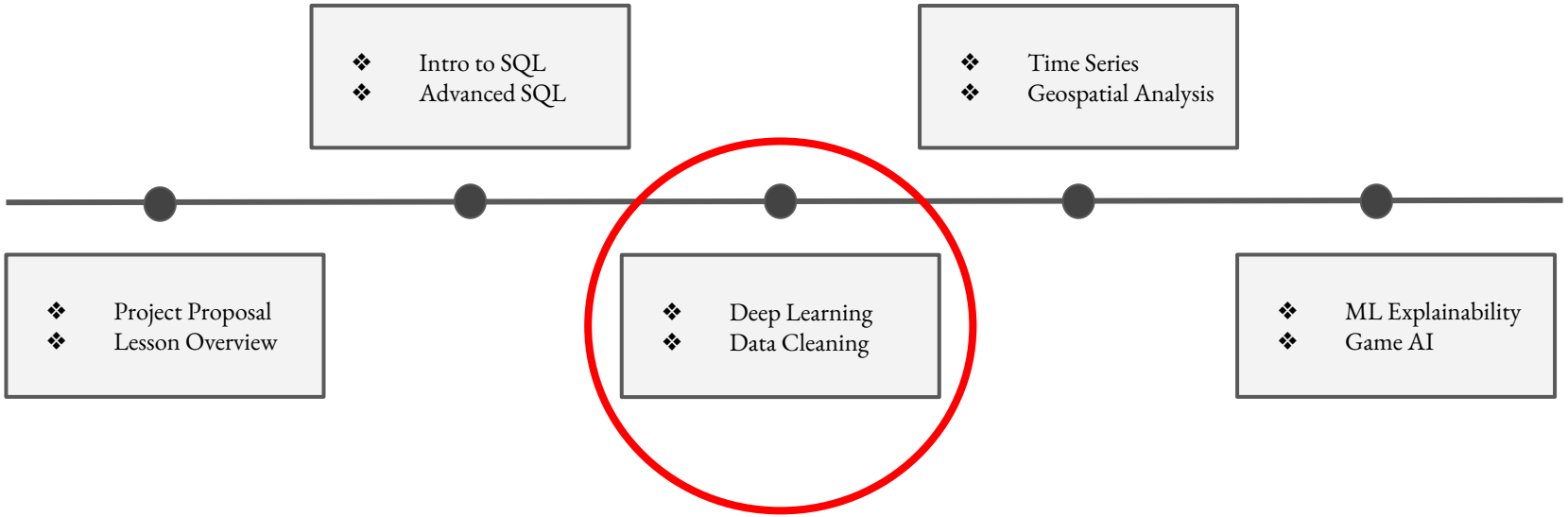
```
star_query = "SELECT * FROM `bigquery-public-data.github_repos.contents`"
show_amount_of_data_scanned(star_query)

basic_query = "SELECT size, binary FROM `bigquery-public-data.github_repos.content
s`"
show_amount_of_data_scanned(basic_query)
```

```
Data processed: 2623.284 GB
Data processed: 2.466 GB
```

Deep Learning & Data Cleaning

이재욱, 이한나, 이나경, 이태경

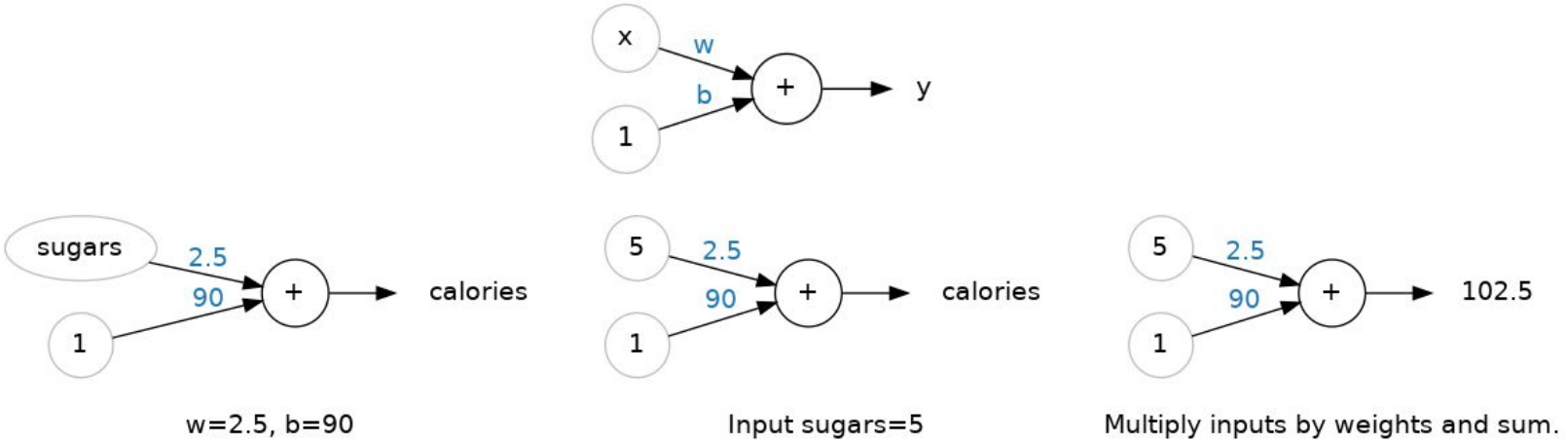
Timeline


Intro to SQL
Advanced SQL
Time Series
Geospatial Analysis
Project Proposal
Lesson Overview
Deep Learning
Data Cleaning
ML Explainability
Game AI

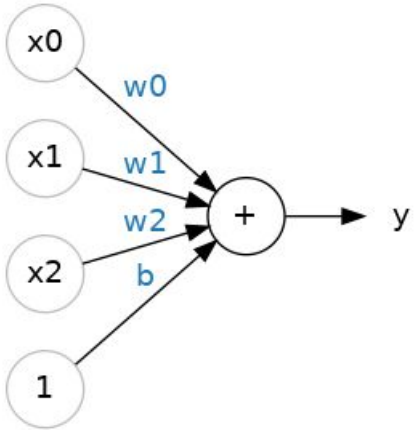
Deep Learning

Approach to Machine Learning characterized by stacks of computations

Neural Networks - Defining model of Deep Learning

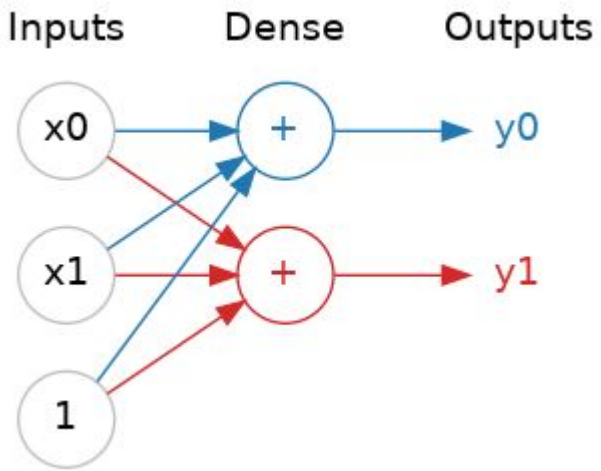


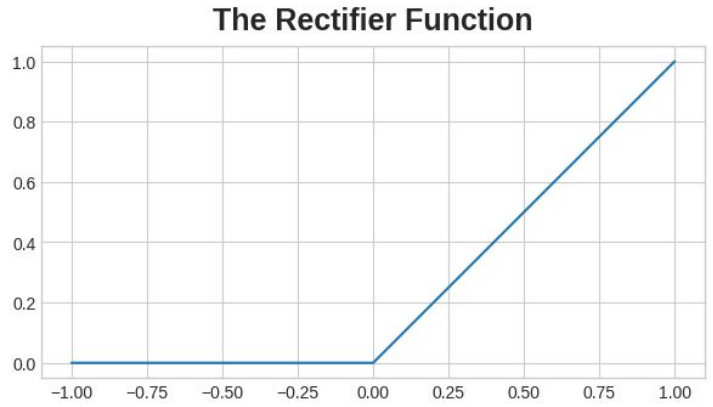
Keras

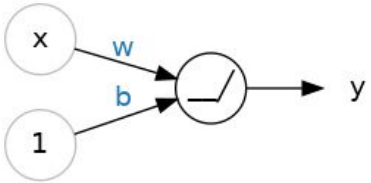



```
from tensorflow import keras
from tensorflow.keras import layers

#Network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1,
input_shape=[3])
])
```

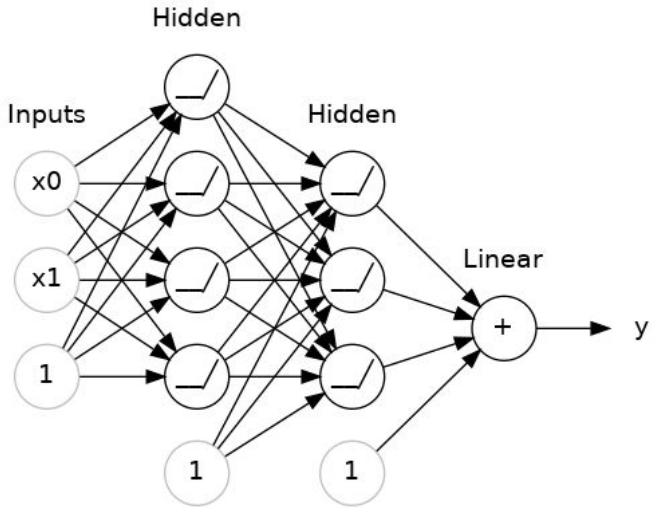
Multiple Layers & Activation Functions





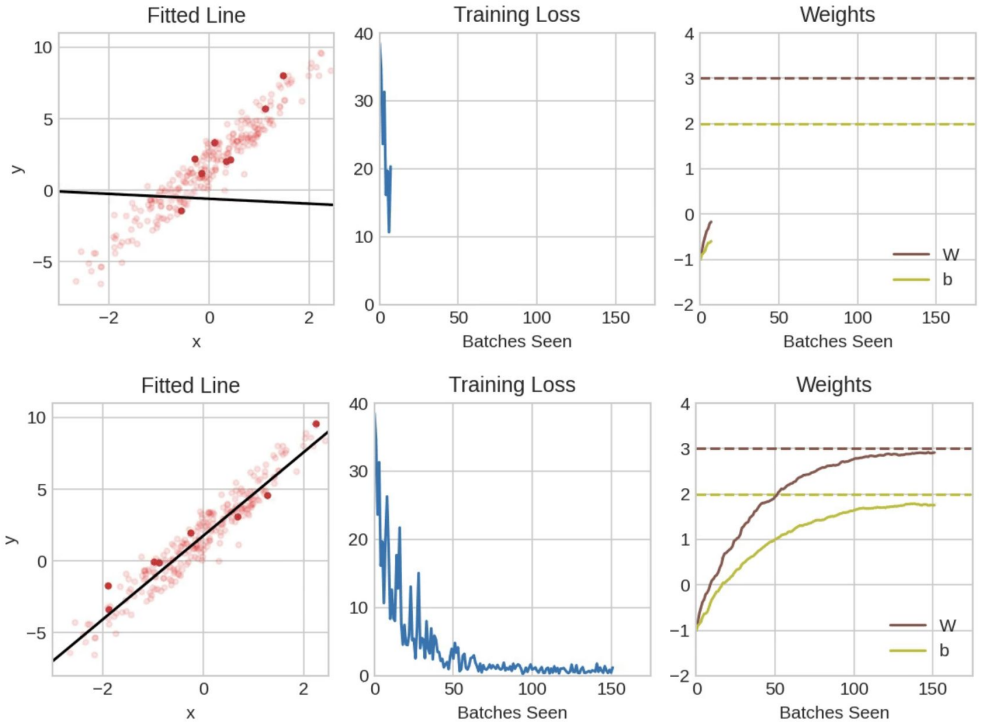


Multiple Layers & Activation Functions



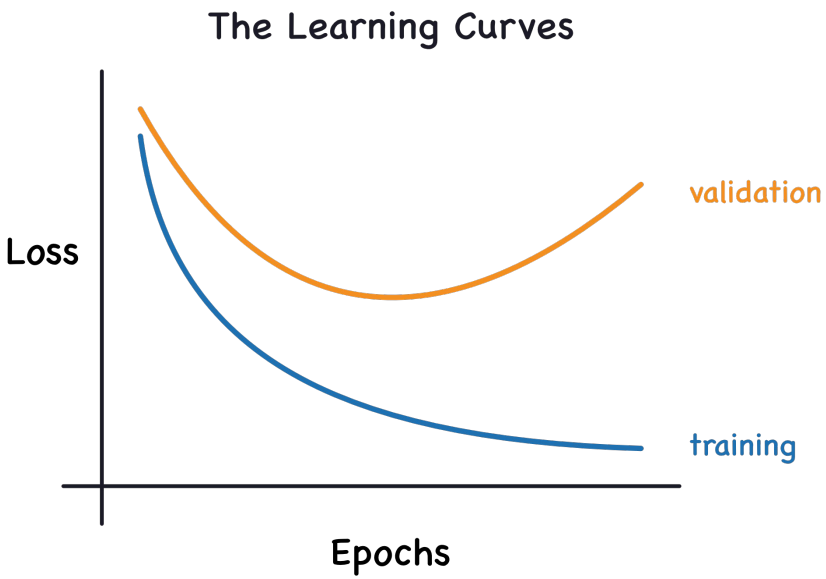

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=4, activation='relu', input_shape=[2]),
    layers.Dense(units=3, activation='relu'),
    # the linear output layer
    layers.Dense(units=1),
])
```

Stochastic Gradient Descent




```
from tensorflow import keras
from tensorflow.keras import layers

model.compile(
    optimizer="adam",
    loss="mae",
  )

history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=256,
    epochs=10,
)
```

Overfitting and Underfitting

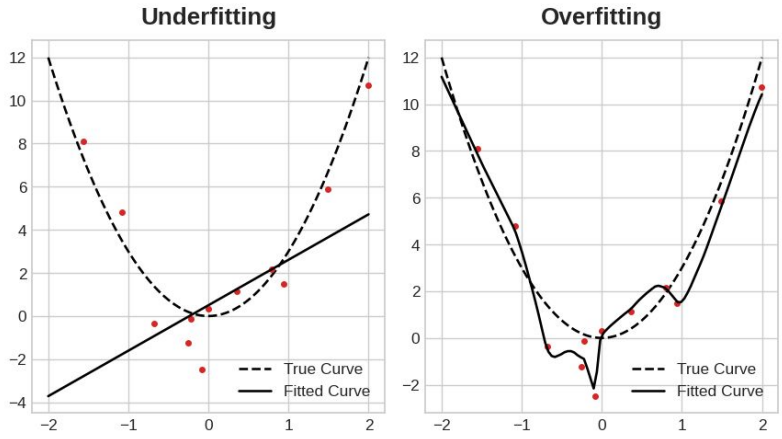


Validation loss gives an estimate of the expected error on data

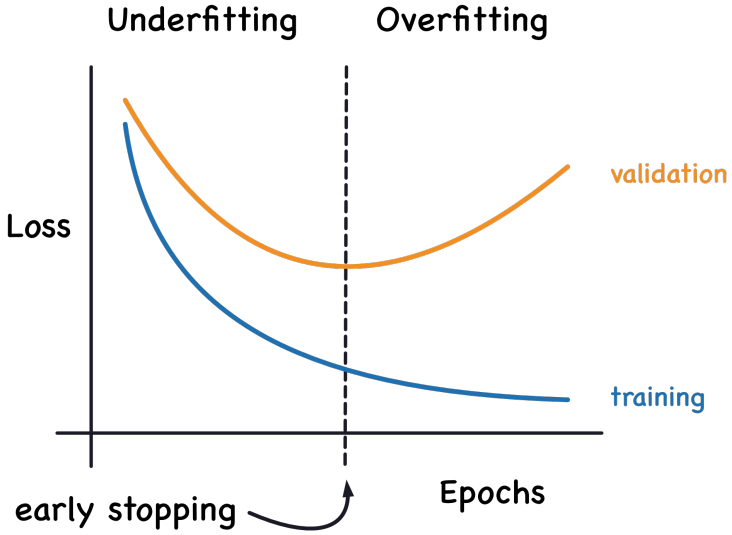
Training Data: **signal** and **noise**

Signal: Generalize to predict new data

Noise: Part that is only true of the training data



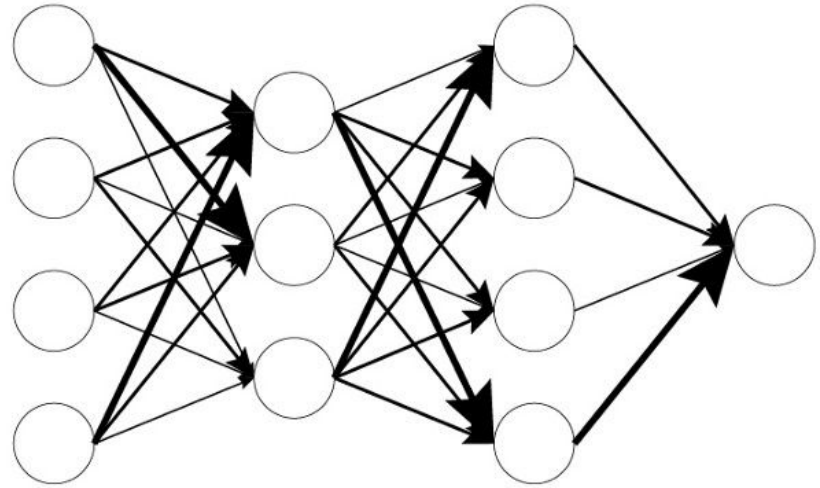
Overfitting and Underfitting




```python
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)
```

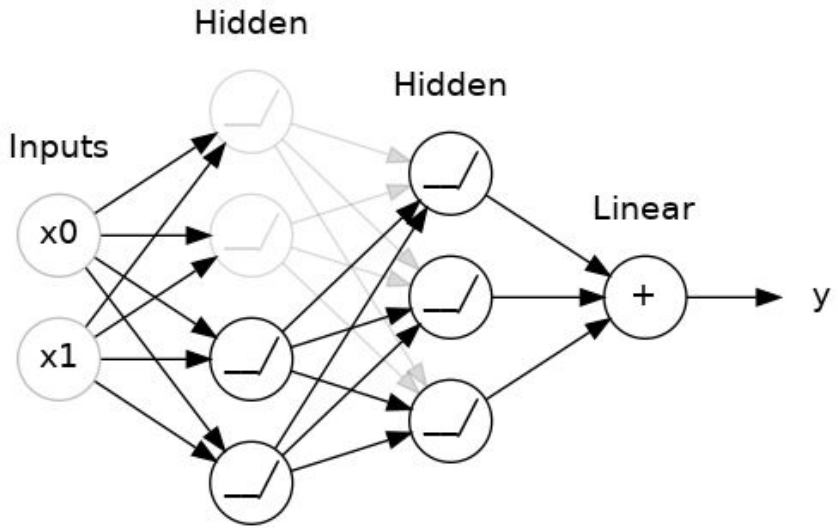
We keep the model where the validation loss is at a minimum

Batch Normalization



- **Raw** signal
- **High interdependancy** between distributions
- **Slow** and **unstable** training

Batch Normalization




```
layers.Dense(16),
layers.BatchNormalization(),
layers.Activation('relu'),
```

Data Cleaning (Handling Missing Values)

```
# get the number of missing data points per column
missing_values_count = nfl_data.isnull().sum()

# look at the # of missing points in the first ten columns
missing_values_count[0:10]
```

```
Date               0
GameID             0
Drive              0
qtr                0
down           61154
time             224
TimeUnder          0
TimeSecs         224
PlayTimeDiff     444
SideofField      528
dtype: int64
```

Is this value missing because it wasn't recorded
or
because it doesn't exist?

From dataset of events that occured in American Football games

Data Cleaning (Handling Missing Values)

Drop Missing Value:

```
# remove all the rows that contain a missing value
nfl_data.dropna()
```

:

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA

From dataset of events that occured in American Football games

Data Cleaning (Handling Missing Values)

Fill Missing Value:

```
# replace all NA's with 0
subset_nfl_data.fillna(0)
```

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Wir
0	2.014474	0.000000	0.000000	0.485675	0.514325	0.546433	0.453567	0.4
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.5
2	-1.402760	0.000000	0.000000	0.551088	0.448912	0.510793	0.489207	0.5
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.5
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.4

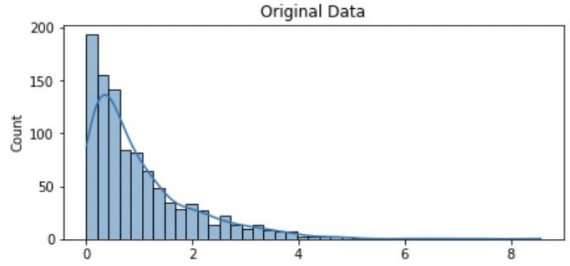
```
# replace all NA's the value that comes directly after it in the same column,
# then replace all the remaining na's with 0
subset_nfl_data.fillna(method='bfill', axis=0).fillna(0)
```

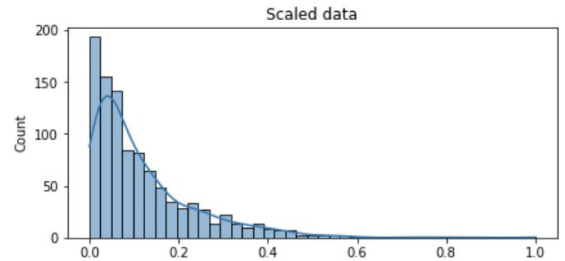
	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Wir
0	2.014474	-1.068169	1.146076	0.485675	0.514325	0.546433	0.453567	0.4
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.5
2	-1.402760	3.318841	-5.031425	0.551088	0.448912	0.510793	0.489207	0.5
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.5
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.4

From dataset of events that occured in American Football games

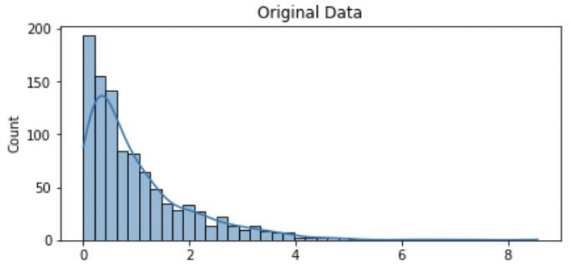
Scaling and Normalization

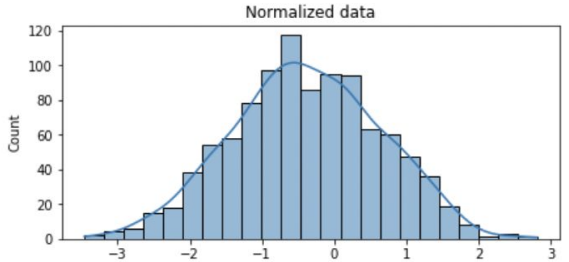
```
# generate 1000 data points randomly drawn from an exponential distribution
original_data = np.random.exponential(size=1000)

# mix-max scale the data between 0 and 1
scaled_data = minmax_scaling(original_data, columns=[0])
```






```
# normalize the exponential data with boxcox
normalized_data = stats.boxcox(original_data)
```

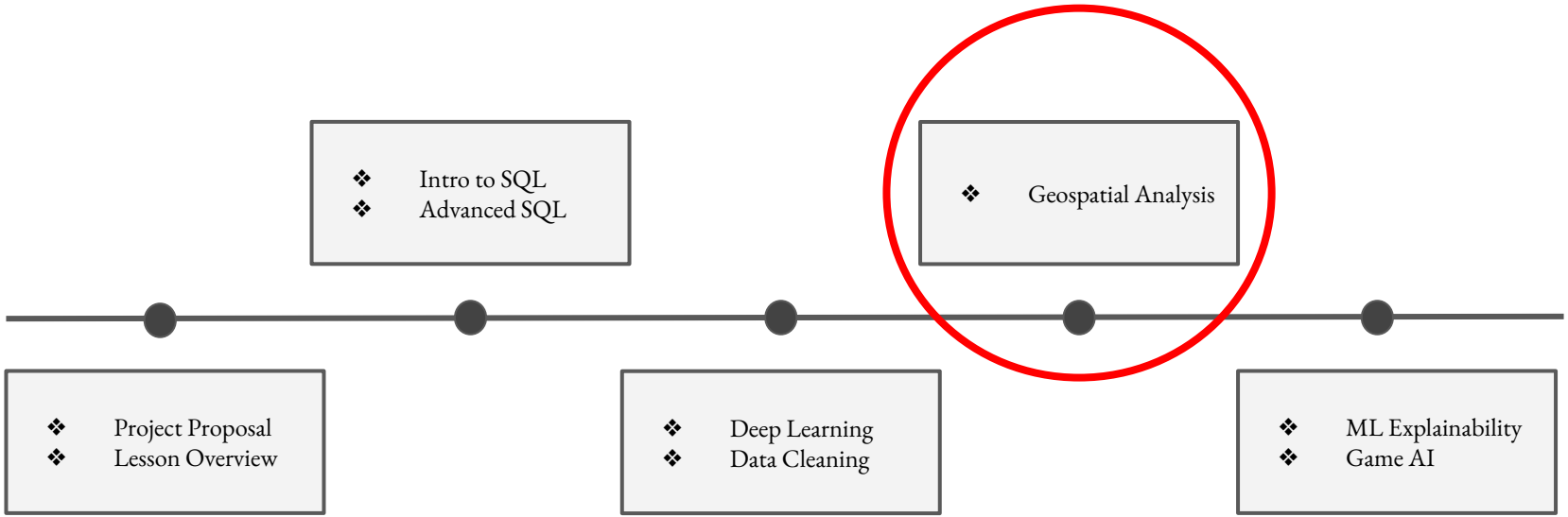




Geospatial Analysis

이재욱, 이한나, 이나경, 이태경

Timeline


Intro to SQL
Advanced SQL
Geospatial Analysis
Project Proposal
Lesson Overview
Deep Learning
Data Cleaning
ML Explainability
Game AI

Geospatial Analysis

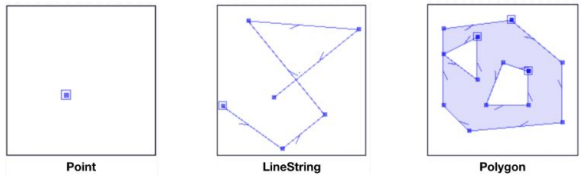
In [1]:
```
import geopandas as gpd
```

- Shapefile
- GeoJSON
- KML
- GPKG

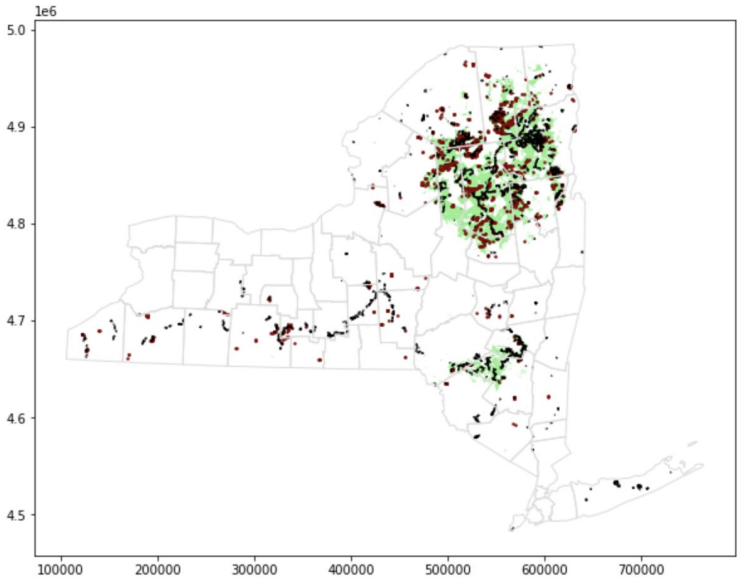
```
full_data = gpd.read_file
```

UPDATE_	OFFICE	ACRES	LANDS_UID	GREENCERT	SHAPE_AREA	SHAPE_LEN
5/12	STAMFORD	738.620192	103	N	2.990365e+06	7927.662385
5/12	STAMFORD	282.553140	1218	N	1.143940e+06	4776.375600
5/12	STAMFORD	234.291262	1780	N	9.485476e+05	5783.070364
5/12	STAMFORD	450.106464	2060	N	1.822293e+06	7021.644833
12/96	RAY BROOK	69.702387	1517	N	2.821959e+05	2663.909932

Creating a Map

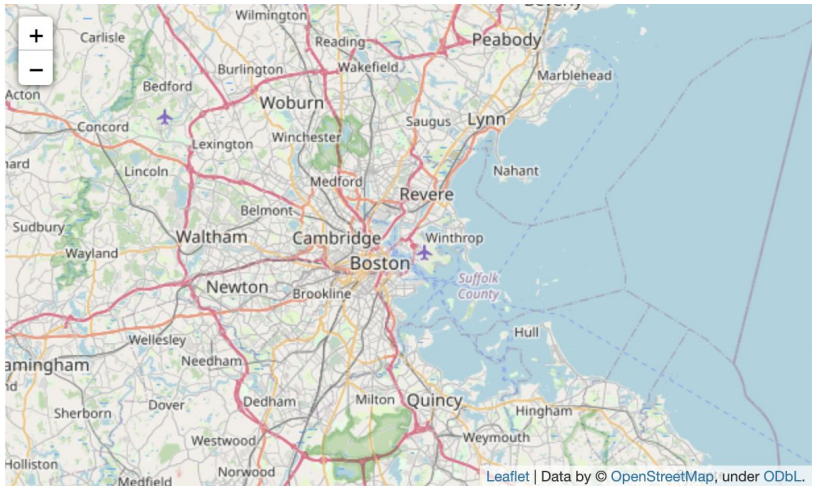



```
# Campsites in New York state (Point)
POI_data = gpd.read_file("../input/geospatial-learn-course-data/DEC_pointsinterest/DEC_pointsinterest/Decptsofinterest.shp")
campsites = POI_data.loc[POI_data.ASSET=='PRIMITIVE CAMPSITE'].copy()

# Foot trails in New York state (LineString)
roads_trails = gpd.read_file("../input/geospatial-learn-course-data/DEC_roadstrails/DEC_roadstrails/Decroadstrails.shp")
trails = roads_trails.loc[roads_trails.ASSET=='FOOT TRAIL'].copy()

# County boundaries in New York state (Polygon)
counties = gpd.read_file("../input/geospatial-learn-course-data/NY_county_boundaries/NY_county_boundaries/NY_county_boundaries.shp")
```

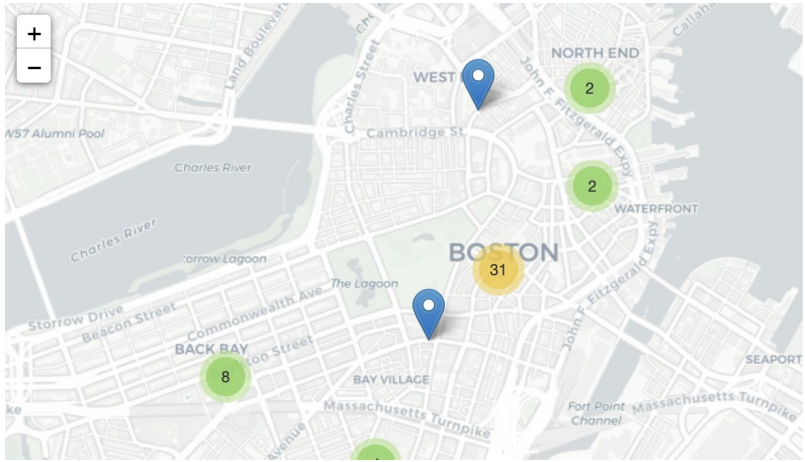


Interactive Maps

```
m_1 = folium.Map(location=[42.32,-71.0589], tiles='openstreetmap', zoom_start=10)
```

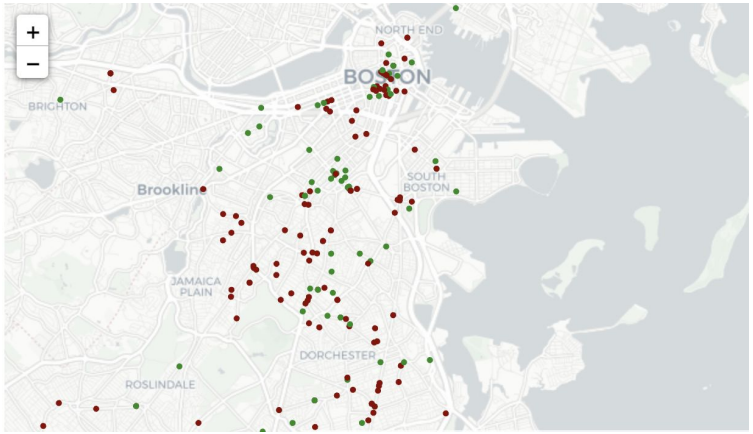



```
m_3 = folium.Map(location=[42.32,-71.0589], tiles='cartodbpositron', zoom_start=13)

# Add points to the map
mc = MarkerCluster()
for idx, row in daytime_robberies.iterrows():
    if not math.isnan(row['Long']) and not math.isnan(row['Lat']):
        mc.add_child(Marker([row['Lat'], row['Long']]))
m_3.add_child(mc)
```

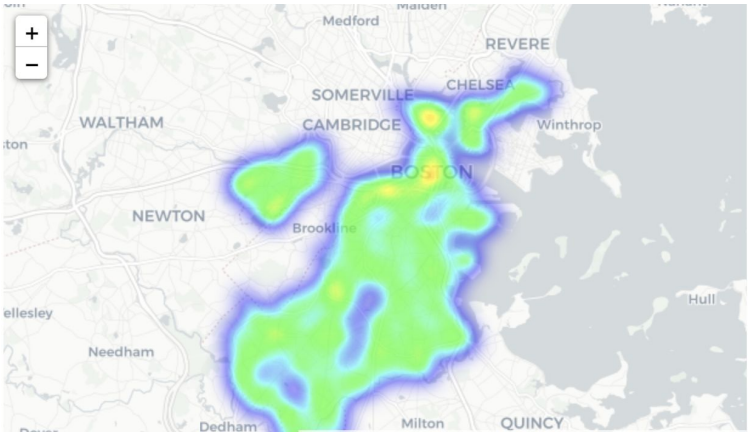


Types of Maps

Bubble Map



Heatmap



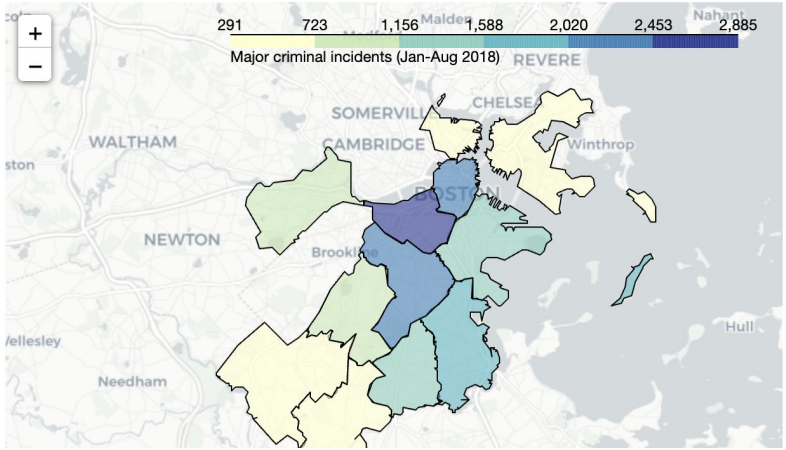
Choropleth Maps

DISTRICT	
A15	MULTIPOLYGON (((-71.07416 42.39051, -71.07415 ...
A7	MULTIPOLYGON (((-70.99644 42.39557, -70.99644 ...
A1	POLYGON ((-71.05200 42.36884, -71.05169 42.368...
C6	POLYGON ((-71.04406 42.35403, -71.04412 42.353...
D4	POLYGON ((-71.07416 42.35724, -71.07359 42.357...

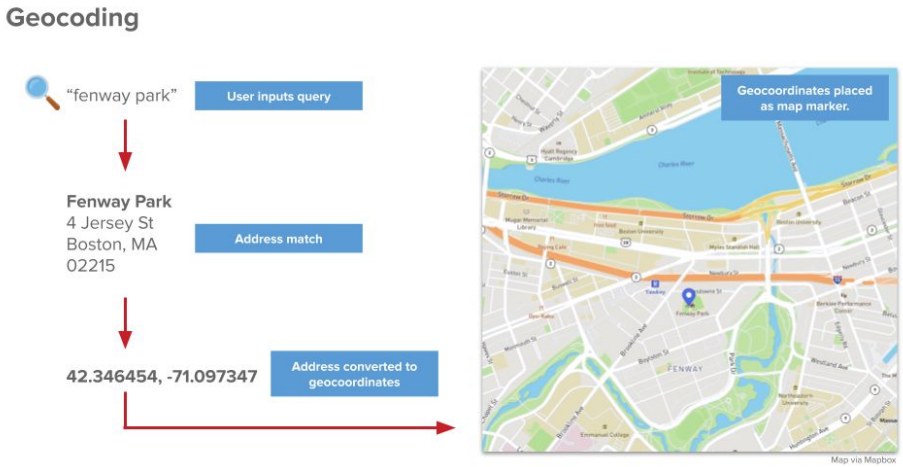
```
plot_dict = crimes.DISTRICT.value_counts()
plot_dict.head()
```

```
D4     2885
B2     2231
A1     2130
C11    1899
B3     1421
Name: DISTRICT, dtype: int64
```

```
m_6 = folium.Map(location=[42.32,-71.0589], tiles='cartodbpositron', zoo
m_start=12)

# Add a choropleth map to the base map
Choropleth(geo_data=districts.__geo_interface__,
           data=plot_dict,
           key_on="feature.id",
           fill_color='YlGnBu',
           legend_name='Major criminal incidents (Jan-Aug 2018)'
          ).add_to(m_6)
```



Manipulating Geospatial Data




```
from geopy.geocoders import Nominatim
```


```
geolocator = Nominatim(user_agent="kaggle_learn")
location = geolocator.geocode("Pyramid of Khufu")

print(location.point)
print(location.address)
```

Latitude:
29.97915995

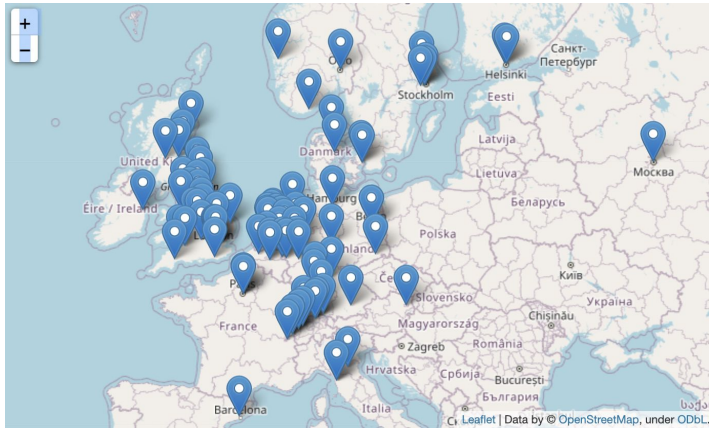
Longitude:
31.134215650388754

Manipulating Geospatial Data

```
universities = pd.read_csv("../input/geospatial-learn-course-data/top_universities.csv")
universities.head()
```

	Name
0	University of Oxford
1	University of Cambridge
2	Imperial College London
3	ETH Zurich
4	UCL

```
def my_geocoder(row):
    try:
        point = geolocator.geocode(row).point
        return pd.Series({'Latitude': point.latitude, 'Longitude': point.longitude})
    except:
        return None

universities[['Latitude', 'Longitude']] = universities.apply(lambda x: my_geocoder(x['Name']), axis=1)

print("{}% of addresses were geocoded!".format(
    (1 - sum(np.isnan(universities["Latitude"])) / len(universities)) * 100))
```



Proximity Analysis

Measuring distance

```
# Select one release incident in particular
recent_release = releases.iloc[360]

# Measure distance from release to each station
distances = stations.geometry.distance(recent_release.geometry)
distances
```

```
print('Closest monitoring station ({} feet):'.format(distances.min()))
print(stations.iloc[distances.idxmin()][["ADDRESS", "LATITUDE", "LONGITUDE"]])
```

```
Closest monitoring station (3780.623590556444 feet):
ADDRESS      3100 Penrose Ferry Road
LATITUDE                   39.91279
LONGITUDE                -75.185448
Name: 9, dtype: object
```

```
print(stations.crs)
print(releases.crs)
```

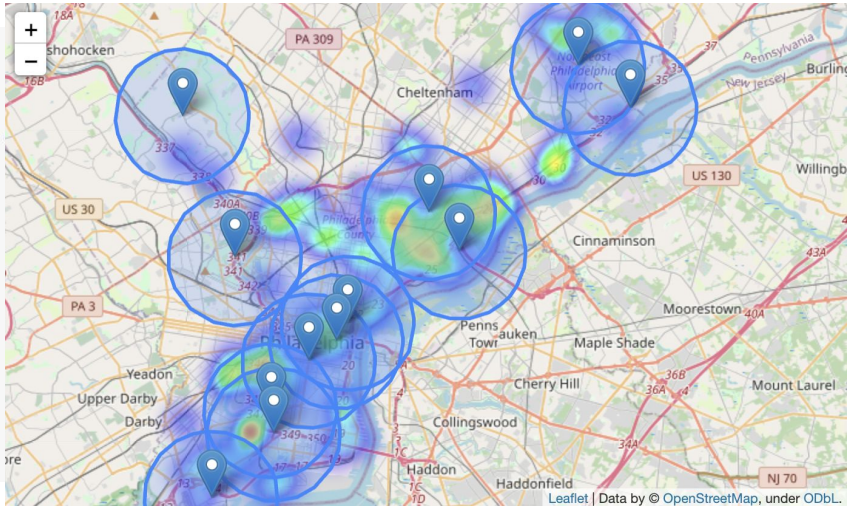
```
epsg:2272
epsg:2272
```

Proximity Analysis

Creating a buffer

```
two_mile_buffer = stations.geometry.buffer(2*5280)
two_mile_buffer.head()
```

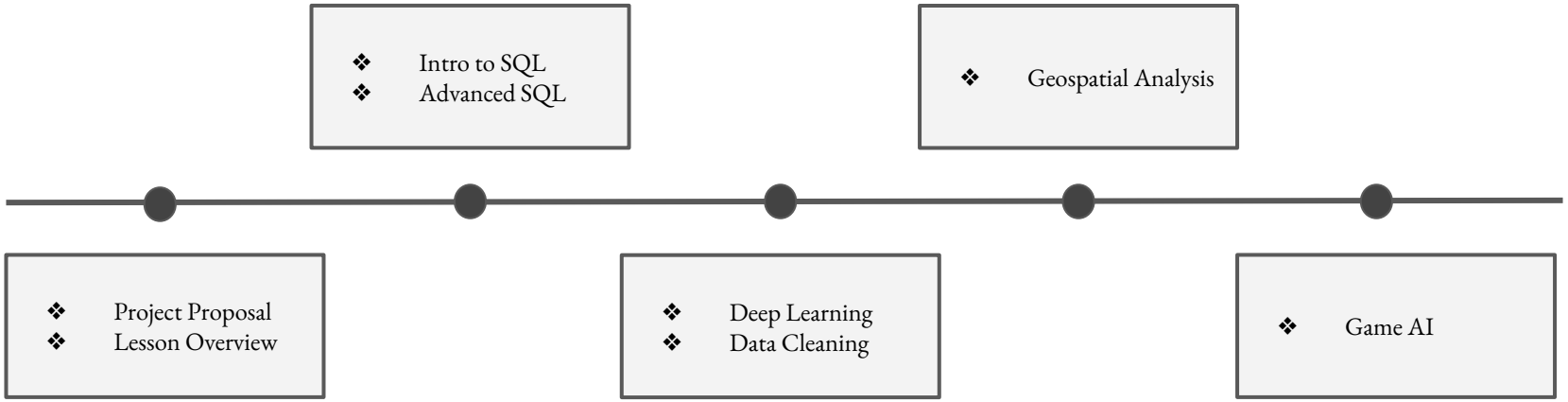
```
# Create map with release incidents and monitoring stations
m = folium.Map(location=[39.9526,-75.1652], zoom_start=11)
HeatMap(data=releases[['LATITUDE', 'LONGITUDE']], radius=15).
for idx, row in stations.iterrows():
    Marker([row['LATITUDE'], row['LONGITUDE']]).add_to(m)

# Plot each polygon on the map
GeoJson(two_mile_buffer.to_crs(epsg=4326)).add_to(m)

# Show the map
m
```



End of Semester Summary

이재욱, 이한나, 이나경, 이태경

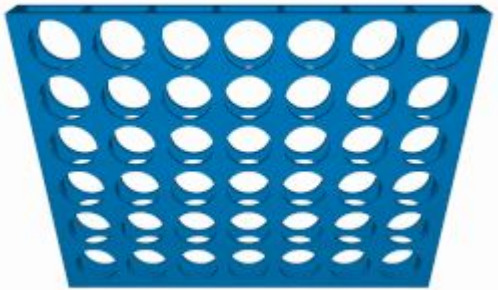
Timeline


❖ Project Proposal
❖ Lesson Overview
❖ Intro to SQL
❖ Advanced SQL
❖ Deep Learning
❖ Data Cleaning
❖ Geospatial Analysis
❖ Game AI

SQL

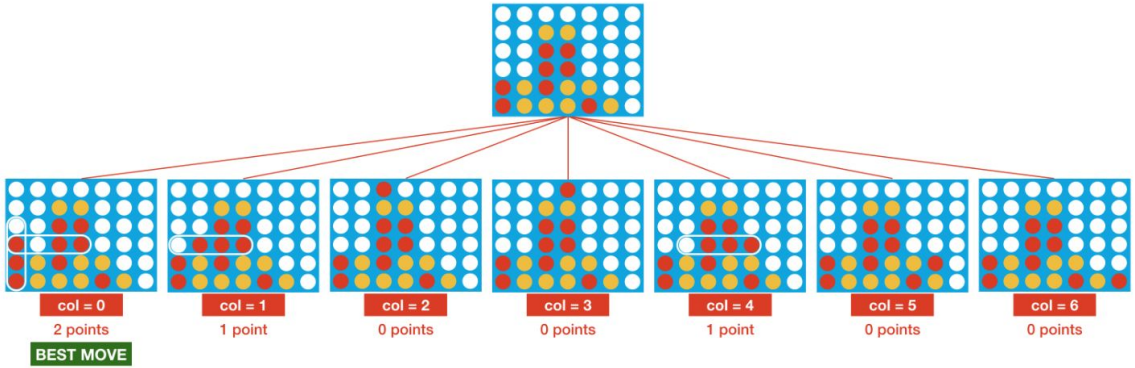
SQL	Python
● Programming language to manage and retrieve data from databases ● Focus is to ‘create’ and ‘maintain’ the datasets ● Access and extract data from database ● Not designed for high level data ● Industry Standard ● Querying data is faster than Python (Simple)	● Analyze and manipulate data ● Specifically designed to create applications

Game AI

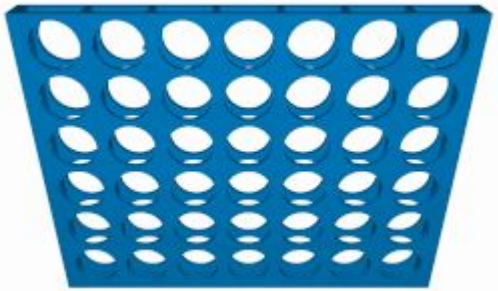


Traditional Method

- Minimax algorithm to dramatically improve your agent
- Use heuristic function
- Agent plans to select move that maximizes score



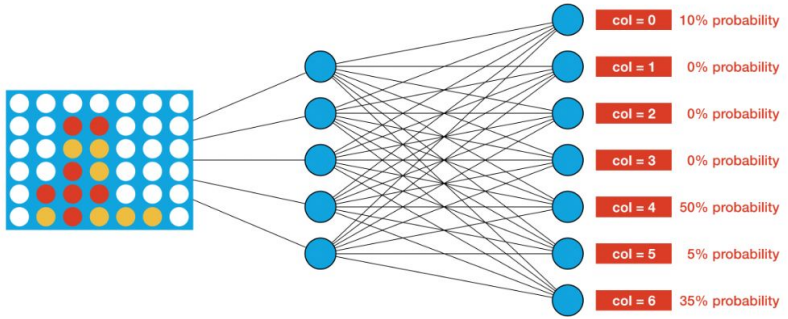
Game AI



```
# Create the game environment
env = make("connectx")

# Two random agents play one game round
env.run([agent1, "random"])

# Show the game
env.render(mode="ipython")
```

Reinforcement Learning

- Advanced techniques for creating intelligent agents
- No heuristic function
- Replace heuristic with neural networks
- Display probability for each possible move

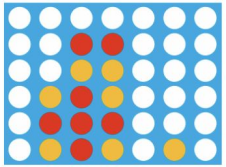


Game AI _ Defining Agents

```
# Selects random valid column
def agent_random(obs, config):
    valid_moves = [col for col in range(config.columns) if obs.board[col] == 0]
    return random.choice(valid_moves)

# Selects middle column
def agent_middle(obs, config):
    return config.columns//2

# Selects leftmost valid column
def agent_leftmost(obs, config):
    valid_moves = [col for col in range(config.columns) if obs.board[col] == 0]
    return valid_moves[0]
```

obs.

- `obs.board` - the game board (a Python list with one item for each grid location)
- `obs.mark` - the piece assigned to the agent (either `1` or `2`)



`obs.board` would be `[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 1, 2, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 1, 2, 0, 2, 0]`.

config.

- `config.columns` - number of columns in the game board (`7` for Connect Four)
- `config.rows` - number of rows in the game board (`6` for Connect Four)
- `config.inarow` - number of pieces a player needs to get in a row in order to win (`4` for Connect Four)