Disclaimer: **These slides are copyrighted and strictly for personal use only**

- This document is reserved for people enrolled into the [Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir]

- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

- All the Best and Happy Learning!

Two major aspects associated with any Project -

1. Writing high performance Spark Code to achieve the required results with Transformations and Performance Tuning. (Technical aspect which is comparatively easy to realise)
2. Gaining the needed Business knowledge that would eventually lead to an efficient solution plan. (Business aspect which is hard to achieve)

Capturing a project requirement / Problem Statement and a highly effective realistic solution plan for the same is crucial and difficult.

Note: It is important to know the following

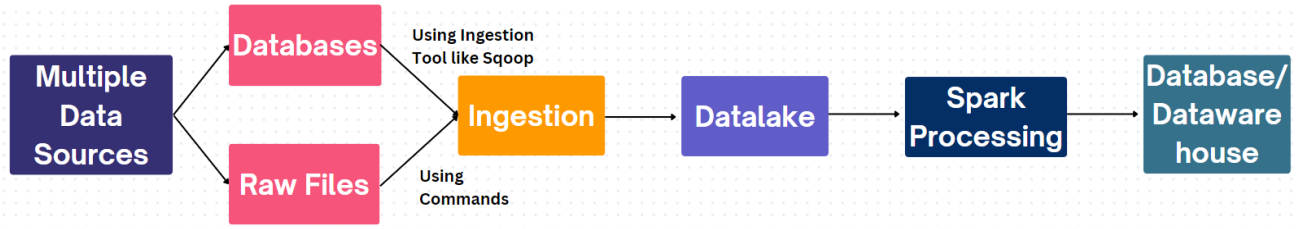
- Industry Standards and Norms of working on a project using Agile Methodology
- Deploying the code to different environments like Dev, Stage, Prod (CICD)
- Data Cleaning Approaches like
    a. Handling duplicates
    b. Dealing with Null values
    c. Handling Missing Values
    d. Changing Datatypes
- Unit Testing using tools like Pytest
- Transformations used in order to enrich the data and bring it to a desired form that can be consumed by the downstream teams.
- About how updates are tackled like - SCD Slowly Changing Dimensions.
- Resource Estimation for the Project.
- Infrastructures used while working on the Project.
- What is the amount of data that is dealt on a day to day basis?
- What is the size of the cluster used?
- About your role in the Big Data Project.

Example Problem Statements of various domains

1. **Reporting :**

Generating Reports that provides 360 view of customers which would help the representatives for pitching in to sell latest products or services and get the deal to a closure.

Example Problem Statement :



Some Pointers :

- What was the need for transferring the data from Database to Datalake when the queries could have been directly executed on the database tables?

Since Databases are majorly used for handling the day to day transactions, applying transformations for analysis purposes could slow down and overburden the databases. Therefore, datalakes come into picture to handle historical data for analysis purposes.

2. **Master Data Management (MDM) :**

MDM is a single source of truth for a company which stores the data belonging to different entities like - customer, vendor, product, location…

Example Problem Statement : Creation of centralized data repository for a company which would involve -

Ingesting the data from many sources -> Cleaning the Data -> Transforming and Enriching the Data -> to provide the latest up-to-date data for report generation

3. **Finance Domain :**

Financial institutes like banks and other alternatives like lending clubs (peer-to-peer platform) require to analyse their customer data to calculate the risk factor involved in approving the loan for a particular customer.

With this analysis of risk-factor score they can take actions like disapproving the loans or increasing the interest rates.

4. **Co-Brand cards :**

A banking related use-cases that involves product analytics to make business decisions.

5. **Healthcare :**

Predictive analytics is carried out to improve patient health outcomes and reduce hospital readmissions.

Data is collected from various sources and devices like

- Electronic health records : which consists of patient's history, treatment and outcomes

- Wearable devices : like Fitbit and Apple watches are used to monitor the health parameters.

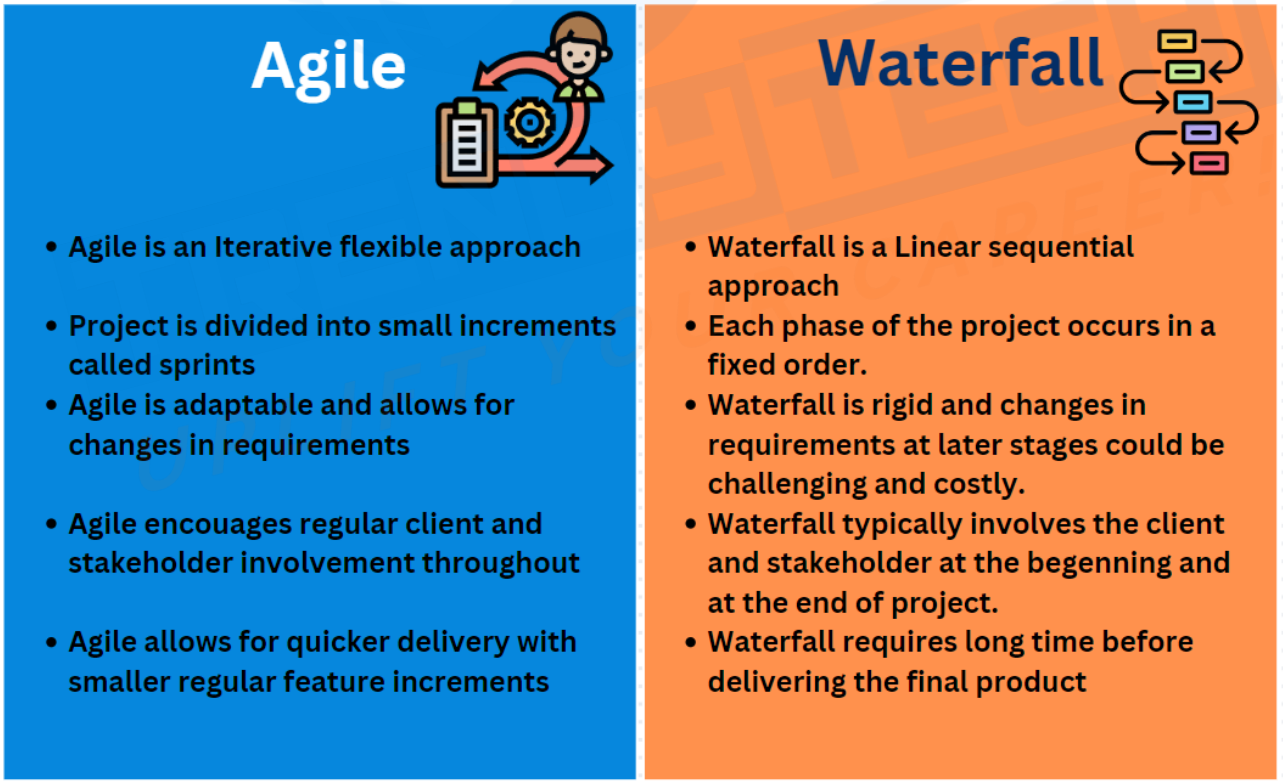
- Genomics Data : Personalized treatments based on the patient's genetics.

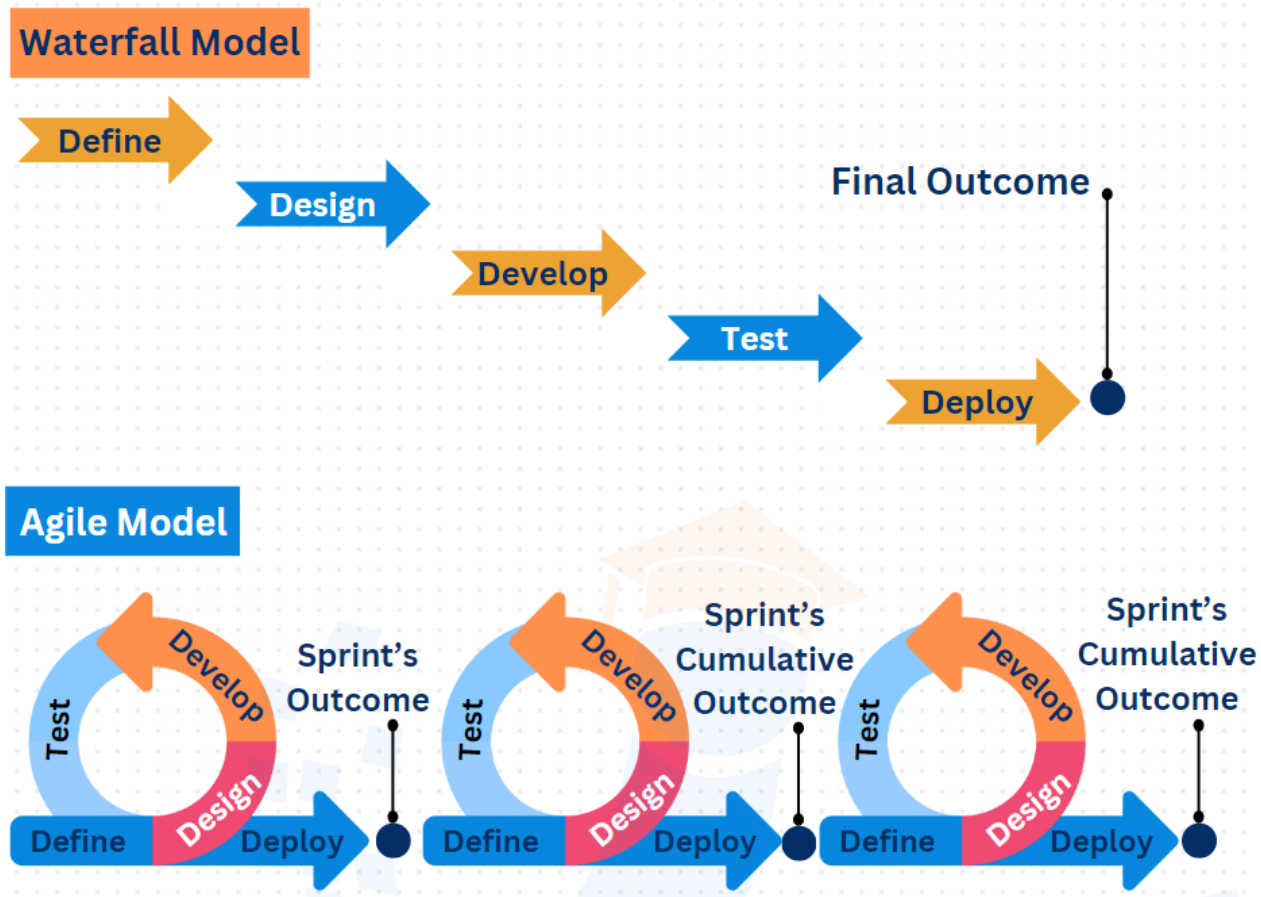
- Social Media Data : Patient Feedback, Experience and Sentiments.

Agile Methodology

Tasks will be assigned in a formal way through a ticketing system while working on a project.

Agile Vs Waterfall Model





Key Points:

- Agile works in Sprints (each sprint can span across 1 to 2 weeks)
- Agile works on the principle of incremental development of project
- With each sprint, new features are added and the product gets enhanced.
- With client or stakeholder feedback in the initial stages itself, major errors regarding the key requirement of the project is avoided.

Dataset creation

Basically we had a very big file and from this file we have created the required datasets i.e. customers_data, loans_data, loan_repayments and loan_defaulters.

Creation of customers_data
The customers_data is having information about the borrowers.

Selecting the columns member_id, emp_title, emp_length, home_ownership, annual_inc, addr_state, zip_code, country, grade, sub_grade, verification_status, tot_hi_cred_lim, application_type, annual_inc_joint, verification_status_joint of the raw file to create customers_data.

Steps for dataset creation
Step1- Using “withColumn”, create a new column “emp_id”.
As the accepted_2007_to_2018Q4.csv dataset is having a member_id column which is having null values, thus we have to create a unique emp_id column.

Below query is used to create the column “emp_id”.
withColumn("emp_id", sha2(concat_ws('||', *['emp_title','emp_length','home_ownership','annual_inc','zip_code','addr_state','grade','sub_grade','verification_status']), 256))

Using the “sha2” function to create the hash values for each customer using the information in 9 existing columns of the dataset and creating an emp_id column. And this hash value will repeat only when the values for all 9 columns are the same.

Note:
1. “SHA-2” (Secure Hash Algorithm 2) is a cryptographic hash function that produces a fixed-size (256 or 512 bits) output. It is used to compute the hash of a given string expression or column.

2. ”concat_ws” is a PySpark SQL function that concatenates multiple input string columns or literals into a single string, using a specified delimiter to separate the values. The function stands for "concatenate with separator".

3. “withColumn function” is used to add a new column or replace an existing column in a DataFrame while keeping the original DataFrame unchanged.

For this we have concatenated 9 columns those column are as follows:
'emp_title' ,'emp_length', 'home_ownership','annual_inc', 'zip_code', 'addr_state', 'grade', 'sub_grade', 'verification_status' .

Step2- Using spark sql query to select the information related to customers and using DataFrame writer API to write the output to the required folder. Using repartition(1) to save all the data in a single file.

```
spark.sql("""select emp_id as member_id, emp_title, emp_length,
home_ownership, annual_inc, addr_state, zip_code, 'USA' as country, grade,
sub_grade,
verification_status, tot_hi_cred_lim, application_type, annual_inc_joint,
verification_status_joint from new_table""").repartition(1) \
.write \
.format("csv") \
.mode("overwrite") \
.option("header", True) \
.option("path", "/user/itv006753/ASH/LendingCLubData/raw/Customer_data") \
.save()
```

Creation of Loans_data
Loans_data is having details about the loans given by the institution.

Selecting the columns Loan_id, member_id, loan_amnt, funded_amnt, term, int_rate, installment, issue_d, Loan_status, purpose, title of the raw file to create Loans_data.

Step for dataset creation:
Using the spark sql query to select the information related to loans and using DataFrame writer API to write the output to the required folder. Using repartition(1) to save all the data in a single file.

```
spark.sql("""select id as loan_id, emp_id as member_id, loan_amnt,
funded_amnt, term, int_rate, installment, issue_d, loan_status, purpose, title
from new_table""").repartition(1) \
.write \
.format("csv") \
.mode("overwrite") \
.option("header", True) \
.option("path", "/user/itv006753/ASH/LendingCLubData/raw/Loan_data") \
.save()
```

Creation of Loan_repayments
Loan_repayments is having details about the loans repayment history. Selecting the columns loan_id, total_rec_prncp, total_rec_int, total_rec_late_fee, total_pymnt, last_pymnt_amnt, last_pymnt_d, next_pymnt_d of the raw file to create Loans_data.

Step for dataset creation
Using spark sql query to select the information related to loans and using DataFrame writer API to write the output to the required folder. Using repartition to save all the data in a single file.

```
spark.sql("""select id as loan_id, total_rec_prncp, total_rec_int,
total_rec_late_fee, total_pymnt,
last_pymnt_amnt, last_pymnt_d, next_pymnt_d from
new_table""").repartition(1) \
.write \
.format("csv") \
.mode("overwrite") \
.option("header", True) \
.option("path",
"/user/itv006753/ASH/LendingCLubData/raw/Loan_repayment_data") \
.save()
```

Creation of Loan_defaulters
Loan_defaulters is the dataset having data about all the defaulters.

Selecting the columns member_id(defaulter), delinq_2yrs, delinq_amnt, pub_rec, pub_rec_bankruptcies, inq_last_6mths, total_rec_late_fee, mths_since_last_delinq, mths_since_last_record of the raw file to create Loans_data.

Step for dataset creation
Using spark sql query to select the information related to loans and using DataFrame writer API we have written the output to the required folder. Using repartition to save all the data in a single file.

```
spark.sql("""select emp_id as member_id, delinq_2yrs, delinq_amnt, pub_rec,
pub_rec_bankruptcies, inq_last_6mths, total_rec_late_fee,
mths_since_last_delinq, mths_since_last_record from
new_table""").repartition(1) \
.write \
.format("csv") \
.mode("overwrite") \
.option("header", True) \
```

.option("path",
"/user/itv006753/ASH/LendingCLubData/raw/Loan_defaulters_data") \
.save()

Cleaning of customers_data

During the cleaning process of "customers_data" modifications were applied to the data initially stored in the raw folder. After processing, the cleaned data was saved in the cleaned folder.

1. To give the correct schema we have explicitly mentioned the schema while creating the customer_raw dataframe. The defined schema is as below

customer_schema = 'member_id string, emp_title string, emp_length string, home_ownership string, annual_inc float, addr_state string, zip_code string, country string, grade string, sub_grade string, verification_status string, tot_hi_cred_lim float, application_type string, annual_inc_joint float, verification_status_joint string'

2. To ensure accurate column names, we utilized the "withColumnRenamed" function. The column names were modified as follows:
“.withColumnRenamed("annual_inc", "annual_income") \
.withColumnRenamed("addr_state", "address_state") \
.withColumnRenamed("zip_code", "address_zipcode") \
.withColumnRenamed("country", "address_country") \
.withColumnRenamed("tot_hi_cred_lim", "total_high_credit_limit") \
.withColumnRenamed("annual_inc_joint", "join_annual_income") “

3. To mention the time when we were processing the data we have created the “ingest_date” column and used the “current_timestamp()” function the query used is given below..
”.withColumn("ingest_date", current_timestamp())”

4. To get rid of duplicate records using below spark sql query unique records are selected and saved to new dataframe.
“spark.sql("select * from Customers where annual_income is not null")”

5. As the “emp_length” column i.e. year of experience is in string format and follows a pattern such as "10 years.". So the non digits values(i.e.(\D)) are replaced using “regexp_replace” function and replaced with blank space(“”)
“.withColumn("emp_length", regexp_replace(col("emp_length"), "(\D)", ""))”

6. To convert “emp_length” column which was in string format to integer format it was casted into integer as below.

“.withColumn("emp_length", customer_income_cleaned.emp_length.cast("int"))”

7. To replace the nulls in the “emp_length” column with the average employment length. First “avg_emp_duration” was calculated and then it was replaced in the “emp_length” column using the below query
“.na.fill(avg_emp_duration, subset=["emp_length"])”

8. To clean the “address_state” column, all values which were having more than 2 characters were replaced with the “NA” and other values remained unchanged, as shown in the following query..
“.withColumn("address_state", when(length(col("address_state")) > 2, "NA").otherwise(col("address_state")))”

9. This cleaned data is stored to the output folder using the DataFrame writer API in csv and parquet format.

Cleaning of loans_data
During the cleaning process of "loans_data" modifications were applied to the data initially stored in the raw folder. After processing, the cleaned data was saved in the cleaned folder.

1. To give the correct schema we have explicitly mentioned the schema while creating the customer_raw dataframe. In a single step only we have changed the column names and mentioned the right datatypes.
The defined schema is as below.

loan_schema = "loan_id string, member_id string, loan_amount float, funded_amnt float, loan_term_months string, interest_rate float, monthly_installment float, issue_date string, loan_status string, loan_purpose string, loan_title string"

2. To mention the time when we were processing the data we have created the “ingest_date” column and used the current_timestamp() function.
”.withColumn("ingest_date", current_timestamp())”

3. The records are dropped if in any one of the columns in the below check column has the null value
columns_to_check = ["loan_amount", "funded_amnt", "loan_term_months", "interest_rate", "monthly_installment", "issue_date", "loan_status", "loan_purpose"]
Using below query:
loan_df_injested.na.drop(subset = columns_to_check)

4. Using “regexp_replace” function “ months” in the loan_term_months column was replaced with the blank space(””). Then it was casted to float using the .cast() function. After this to convert this month into year this value was divided by 12 and at the end casted into integer using the below query.

```
withColumn("loan_term_months", ((regexp_replace("loan_term_months", " months", "")).cast("float")/12).cast("int")) \
.withColumnRenamed("loan_term_months", "loan_term_years")
```

5. To rename the unnecessary “loan_purpose” with others “when clause” was used. For this the purposes beyond the defined list(loan_purpose_lookup) were renamed to “others” using the below query.

```
loan_purpose_lookup = ["debt_consolidation", "credit_card", "home_improvement", "other", "major_purchase", "medical", "small_business", "car", "vacation",
                       "moving", "house", "wedding", "renewable_energy", "educational"]

.withColumn("loan_purpose",
when(col("loan_purpose").isin(loan_purpose_lookup),
col("loan_purpose")).otherwise("other"))
```

6. This cleaned data is stored to the output folder using the DataFrame writer API in csv and parquet format.

Cleaning of Loan_repayments

During the cleaning process of "Loan_repayments," modifications were applied to the data initially stored in the raw folder. After processing, the cleaned data was saved in the cleaned folder.

1. We ensured the accurate schema by explicitly specifying it while creating the customer_raw dataframe. In one step, we both renamed the columns and assigned the correct data types.

The defined schema is as below.

```
loan_repay_schema =  "loan_id string, total_principle_received float, total_interest_received float, total_late_fee_received float, total_payment_received float, last_payment_amount float, last_payment_date string, next_payment_date string"
```

2. To mention the time when we were processing the data we have created the “ingest_date” column and used the current_timestamp() function.

```
”.withColumn("ingest_date", current_timestamp())”
```

3. Records are discarded if any of the columns listed below contain null values.

columns_to_check = ["total_principle_received", "total_interest_received", "total_late_fee_received", "total_payment_received", "last_payment_amount"]
Using below query:
df.na.drop(subset = columns_to_check)

4. If the total_payment_received is 0.0 but total_principle_received is not 0.0, indicating that the principle has been paid, the situation arises where the total payment received is inexplicably zero. In such cases, we replace total_payment_received with the sum of total_principle_received, total_interest_received, and total_late_fee_received using a "when" clause.

Using the below query this is achieved.
```
.withColumn("total_payment_received", when(
    (col("total_payment_received") == 0.0) & (col("total_principle_received") != 0.0),
col("total_principle_received")+col("total_interest_received")+col("total_late_fee_received"))

.otherwise(col("total_payment_received")))
```

5. To remove the columns where total_payment_received is zero, we selected all the records where total_payment_received is not zero. Below filter condition is used to filter the records and then saved into another dataframe.

```
..filter("total_payment_received != 0.0")
```

6. Since last_payment_date cannot be 0, it must either contain a valid date or a null value. To replace null values, a "when clause" is employed, utilizing the following query.

```
.withColumn("last_payment_date", when(
    (col("last_payment_date") == 0.0),
None).otherwise(col("last_payment_date"))
```

7. Similarly as next_payment_date cannot be 0, either it should be some date or null value. Thus to replace null values ‘when clause’ is used and the query used is as below.

```
.withColumn("next_payment_date", when(
    (col("next_payment_date") == 0.0),
None).otherwise(col("next_payment_date"))
```

8.This cleaned data is stored to the output folder using the DataFrame writer API in csv and parquet format.

Cleaning of Loan_defaulters
While cleaning of Loan_defaulters following changes have been made on the data which was kept in the raw folder and then after processing saved in the cleaned folder.

1. To give the correct schema we have explicitly mentioned the schema while creating the customer_raw dataframe to mention the right datatypes.

The defined schema is as below.
defaulter_schema = "member_id string, delinq_2yrs float, delinq_amnt float, pub_rec float, pub_rec_bankruptcies float, inq_last_6mths float, total_rec_late_fee float, mths_since_last_delinq float, mths_since_last_record float"

In the above delinq_2yrs column which was in string initially was converted into the float datatype so any non-float values were converted into nulls.

2. Using below query delinq_2yrs which was earlier casted to float was converted into integer as time period(i.e.years) should be an integer. And all the nulls were replaced by zero as the time period(i.e.years) cannot be null either it should be some integer or 0. The query used is as follows.
.withColumn("delinq_2yrs", col("delinq_2yrs").cast("integer")).fillna(0, subset = ["delinq_2yrs"])

3. The data was stored into 2 separate files
a.The first dataset contains details of customers who missed or delayed payments, based on the conditions delinq_2yrs > 0 or mths_since_last_delinq > 0. This dataset focuses on delinquency and includes the following columns: member_id, delinq_2yrs, delinq_amnt, int(mths_since_last_delinq). The records were filtered using the query:

spark.sql("select member_id, delinq_2yrs, delinq_amnt, int(mths_since_last_delinq) from loan_defaulter where delinq_2yrs>0 or mths_since_last_delinq>0")

b. In the second dataset information about only those member_id (borrowers) is considered for whom either there is public record(pub_rec) or bankruptcy record (pub_rec_bankruptcies) or enquiries in the last 6 months (inq_last_6mths). This dataset is basically related to public records and the enquiries.

4. Both these cleaned dataset are stored to the output folder using the DataFrame writer API in csv and parquet format.