[Return to Classroom](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)

Optimizing Public Transportation

|  |  |  |
| --- | --- | --- |
| **[REVIEW](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)** | **[CODE REVIEW  10](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)** | **[HISTORY](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)** |

**Requires Changes**

3 specifications require changes

Excellent! You do well. Don't disturb yourself with my observations here I am sure you will resolve it shortly.  
 only little add offset to consumer .py broker to determine the right offset to  
"auto.offset.reset": "earliest" if offset\_earliest else "latest"  
we adjust offset to guarantee to consume specific topic from producers if this first time to consume  
data or there is only one user you will set it earliest else you need to get topics after the last  
topic is consumed.  
you need not to seek for partition to get offset you need adjust offset to the beginning.  
from confluent\_kafka import Consumer, OFFSET\_BEGINNING  
partition.offset = OFFSET\_BEGINNING

 nothing here will replace {station*name}, so this condition will not success  
so you would better remove it from the topic and you need to choose arrival topic you choose  
` elif 'station*' in message\_topic: `  
you need to use the default topic from server.py or change it to yours.  
topic in server.py and faust\_stream.py should be the same  
I enjoy reviewing your submissions thanks.  
Don't forget to give review feedback 

**Additional Resources**

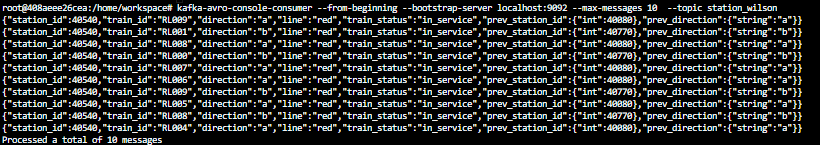
[KSQL aggregate-functions](https://docs.ksqldb.io/en/latest/reference/#aggregate-functions)  
 [Faust userguide](https://faust.readthedocs.io/en/latest/userguide/)  
 [Official REST API Documentation](https://docs.confluent.io/current/connect/references/restapi.html)  
 [Python fastavro Library](https://fastavro.readthedocs.io/en/latest/index.html)  
 [Apache Avro Specification](https://avro.apache.org/docs/1.8.2/spec.html#Maps)  
 [confluent-kafka-python/librdkafka ConfigurationOptions](https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md)  
[Offset managment](https://www.learningjournal.guru/courses/kafka/kafka-foundation-training/offset-management/)

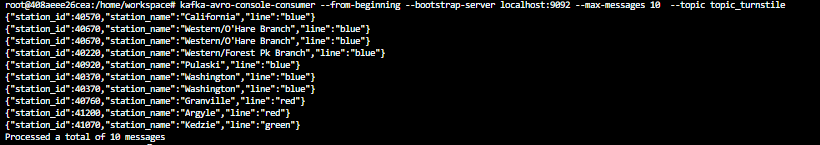
**Kafka Producer**

**Using the Kafka Topics CLI, topics appear for arrivals on each train line in addition to the turnstiles for each of those stations.**

Good job setting up the required topic names for each producer.  
 topic names are consistent and meaningful.  
Your naming is readable meaningful easy to understand  
You choose good names not just traditional names  
 topics appear for arrivals one each train line.  
you need to change the topic to be a dynamic change for each station like {station-name}.arrival topic for each station not only one topic contains all stations  
 replication and partition settings are not simply set to the default.  
You set number of replications and partitions not using default values as in real live projects you need large numbers of replications and partitions.  
Comments: you may think that naming is not important, guess what choosing bad names will make you lost and revision or debugging can be nightmare to you

**Using the Kafka Topics CLI, messages continuously appear for each station on the train line, for both arrivals and turnstile actions.**

All turnstiles and arrivals messages are correctly produced. Well done!  
 messages continuously appear for each station on the train line for arrivals actions.  


 messages continuously appear for each station on the train line for turnstile actions.  
  
 all expected fields are present in the messages.

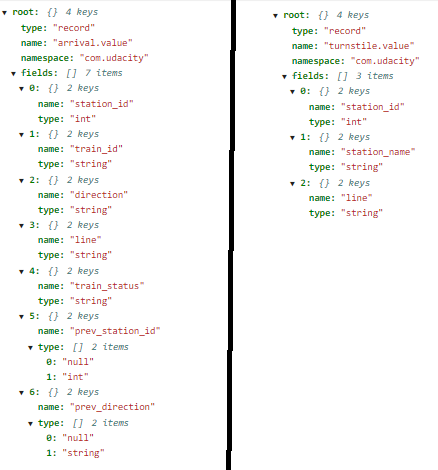
**Comments:**

Those two topics is the base that all the project depend on them any error in them will make influence in all other parts

**Using the Schema Registry API, a schema is visible for arrivals and turnstile events.**

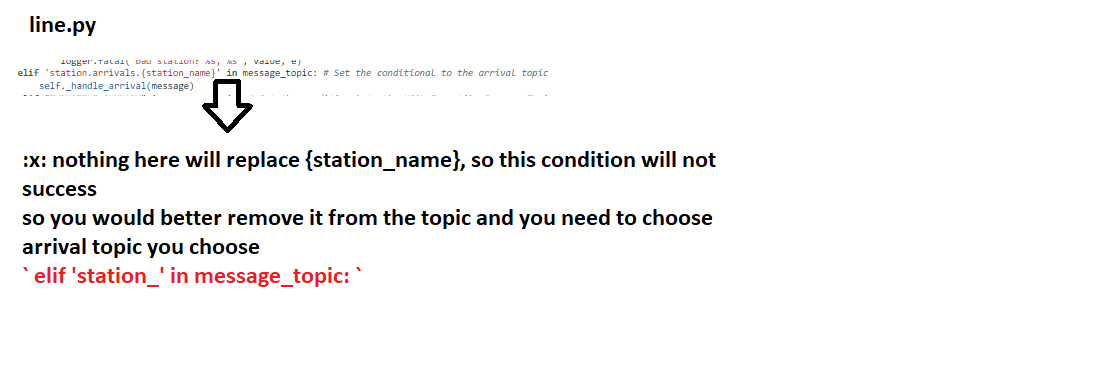
Turnstile/Arrival schemas are correctly defined.  
 schema is visible for arrivals and turnstile events  
 all the required fields are produced and have the appropriate types as defined in the solution schema.

**Comments :**

using appropriate data type is very important this schema is the way producers produce topic and the way consumers consume the topic been produced like a map so It is smart thinking to begin with schema then everything else.  


**Kafka Consumer**

**Stations, status, and weather data appear and update in the Transit Status UI.**

 Good job using the given handler to process a message  
 Good job bootstrap properties appropriately set.  
 Good job using on*assign callback for topic subscription  
 nothing here will replace {station\_name}, so this condition will not success  
so you would better remove it from the topic and you need to choose arrival topic you choose  
` elif 'station*' in message\_topic: `  


**comments :**

A message handler consists of one or multiple filters.  
Each filter must return True for a certain message in order for a message handler  
to become eligible to consume that message.

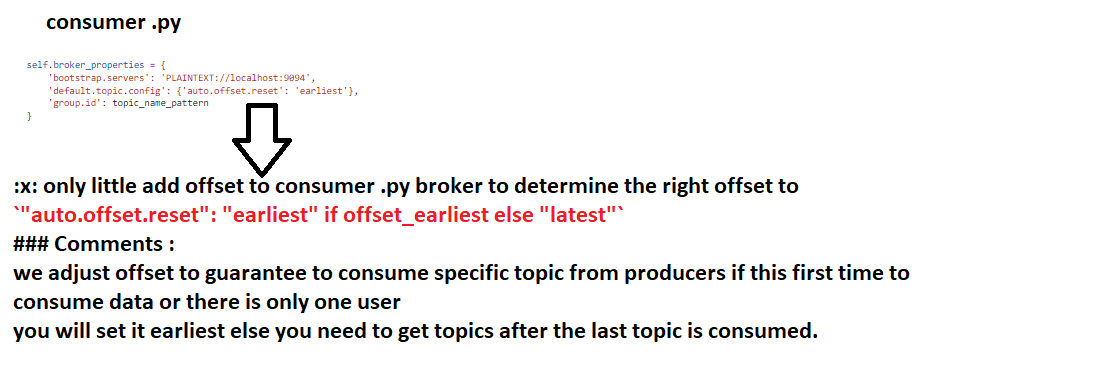
else:

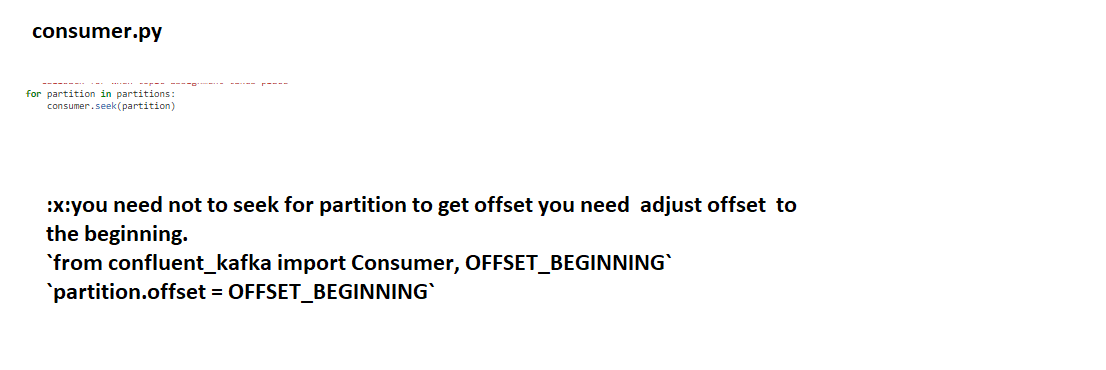
self.message\_handler(message)

logger.info(f"Consumer Message key :{message}")

return 1

**All Blue, Green, and Red Line stations appear in the Transit Status UI.**

 only little add offset to consumer .py broker to determine the right offset to  
"auto.offset.reset": "earliest" if offset\_earliest else "latest"  
we adjust offset to guarantee to consume specific topic from producers if this first time to consume  
data or there is only one user you will set it earliest else you need to get topics after the last  
topic is consumed.  


you need not to seek for partition to get offset you need adjust offset to the beginning.  
from confluent\_kafka import Consumer, OFFSET\_BEGINNING  
partition.offset = OFFSET\_BEGINNING  


**Comments :**

we adjust offset to guarantee to consume specific topic from producers if this first time to consume data or there is only one user  
you will set it earliest else you need to get topics after the last topic is consumed.

**Additional resources**

[Offset managment](https://www.learningjournal.guru/courses/kafka/kafka-foundation-training/offset-management/)

**Kafka REST Proxy**

**Using the kafka-console-consumer, weather messages are visible in the weather topic and are regularly produced as the simulation runs.**

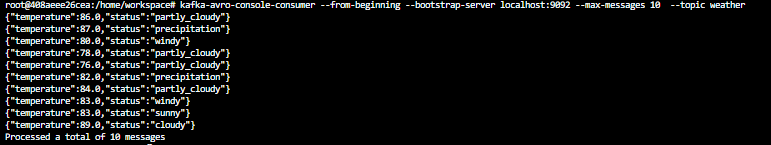
 The weather messages can be properly consumed from the console.  
 both temperature and status are included.  
 Status description is in string format.

**Comments:**

due to rubric temperature suppose not to be string, and status at the same time not supposed to be number , in some cases we find temp:37 status:3 where 3 refer to windy for example.  
Although some student solve this problem by coding it is better to be done in schema itself.

**additional resources**

[Official REST API Documentation](https://docs.confluent.io/current/connect/references/restapi.html)



**Using the Kafka Schema Registry REST API, a schema is defined for the weather topic.**

Awesome: Great Content-Type header is set to the Avro type.  
headers={"Content-Type": "application/vnd.kafka.avro.v2+json"}  
 Great you define weather schema correctly

**Comments:**

Although due to rubric status supposed to be in string format some student make data type string.  
It is better to use enum to ensure no wrong data in weather status here some information about enumerators

* Declaration:Enum, short for enumerated,"" is a data type that consists of predefined values.  
  A constant or variable defined as an enum can store one of the values listed in the enum declaration.
* [enum in lesson 3.12](https://classroom.udacity.com/nanodegrees/nd029/parts/cfa24d55-3ab1-4410-8a26-01d9a568564e/modules/fd1d9ba6-f136-40ff-9f7c-cbbc799a4744/lessons/93ae36b9-54a2-4275-b913-10313fc8f27c/concepts/2b4f6a52-e905-41de-aef1-e2887ee3f799)
* [complex schema avro](https://avro.apache.org/docs/1.8.2/spec.html#schema_complex)



**Kafka Connect**

**Using the kafka-console-consumer, all stations defined in Postgres are visible in the stations topic.**

 all stations defined in Postgres are visible in the stations topic  
 Great all 230 stations are here.

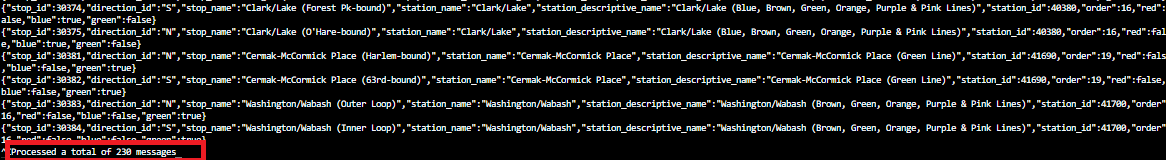
**Comments:**

We here ensure that connection done correctly and all stations (230) are read from DB .  
all connections parammeter define correctly in connector

"connection.url": "jdbc:postgresql://localhost:5432/cta",

"connection.user": "cta\_admin",

"connection.password": "chicago",



**Using the Kafka Connect REST API, the Kafka Connect configuration is configured to use JSON for both key and values.**

**Using the Schema Registry REST API, the schemas for stations key and value are visible.**

Kafka Connect that the key and value are both set to type Avro  
Kafka Connect configuration is configured to use JSON for both key and values.  
headers={"Content-Type": "application/json"}

**comments :**

key.converter  
Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka.  
This controls the format of the keys in messages written to or read from Kafka, and since this is independent of connectors  
it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro.  
value.converter  
Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format  
of the values in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work  
with any serialization format. Examples of common formats include JSON and Avro.

**Additional resources**

[Kafka Connect configuration](https://docs.confluent.io/platform/current/installation/configuration/connect/index.html)

**Using the Kafka Connect REST API, the Kafka Connect configuration uses an incrementing ID, and the ID is configured to be stop\_id.**

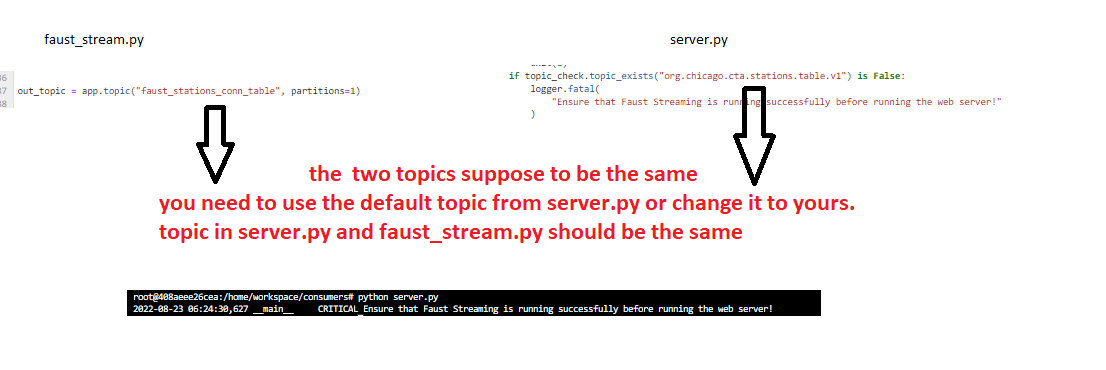
Kafka Connect configuration uses an incrementing ID.  
 the ID is configured to be stop\_id .

**Comments :**

o incrementing: use a strictly incrementing column on each table to detect only new rows. Note that this will not detect modifications or deletions of existing rows.  
o The name of the strictly incrementing column to use to detect new rows. Any empty value indicates the column should be autodetected by looking for an auto-incrementing column. This column may not be nullable.

**Faust Streams**

**A consumer group for Faust is created on the Kafka Connect Stations topic.**

you need to use the default topic from server.py or change it to yours.  
topic in server.py and faust\_stream.py should be the same  
  
connect station topic correctly created.  
 ingesting data from the Kafka Connect stations correctly  
 Faust is pulling data from the correct topic.

**Comments:**

Here It is important to know the relation between connector and faust you need to ingest data from connector use the  
"topic.prefix": then add stations to use in Faust.  
then you can pull data from the output topic directly  
Note: if you use a new topic or user-defined topic different from the topic used in server.py you need to update the topic to be the same  
as a topic in server.py and oufaust\_stream.py should be the same.

**additional resources**

 [Faust userguide](https://faust.readthedocs.io/en/latest/userguide/)

**Data is ingested in the Station format and is then transformed into the TransformedStation format.**

 Great Station format and is then transformed into the TransformedStation format  
 conditional statement to translate the red/green/blue Booleans to “line” in “TransformedStation”

**Comments:**

Using Faust, transform input Station records into TransformedStation records. Note that  
"line" is the color of the station. So if the Station record has the field red set to true,  
then you would set the line of the TransformedStation record to the string "red" same for blue and green.

**suggessions :**

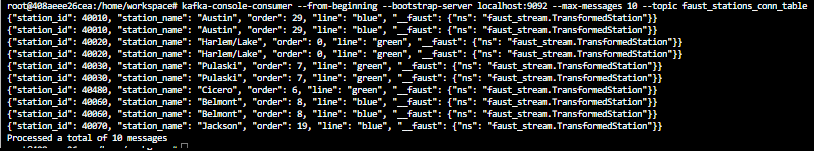
You can only use one line  
line = "red" if station.red else "blue" if station.blue else "green"

**A topic is present in Kafka with the output topic name the student supplied. Inspecting messages in the topic, every station ID is represented.**

 Great You using station ID as the table key  
 they are setting it to the transformed station model.

**Comments:**

You do great by transforming data but due to rubric . you need to add those data in the table not sending them as topics due to the rubric and use station as Table ID that contain transformed data.  
We use table as we will only have the latest station and when another station comes up for the same id it will overwrite the old one. Then we will have latest record for each station.



**KSQL**

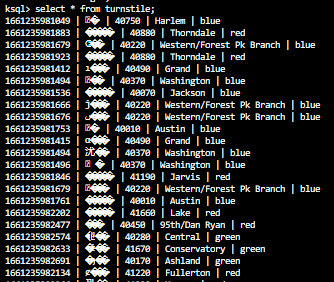
**Using the KSQL CLI, turnstile data is visible in the table TURNSTILE.**

 TURNSTILE table create correctly

**Comments:**

In Ksql statement you need to make station id as key and choose suitable turnstile topic that been configured inproducers/turnstile.py don’t forget that the format is AVRO here

**Additional Resources**

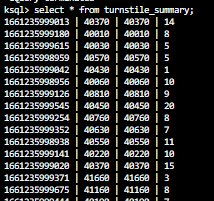
[KSQL aggregate-functions](https://docs.ksqldb.io/en/latest/reference/#aggregate-functions)  


**Using the KSQL CLI, verify that station IDs have an associated count column.**

 TURNSTILE\_SUMMARY table create correctly  
 Table fields being produced to two columns, station\_id, and count.

**Comments:**

After creating Turnstile table then from its result we create TURNSTILE\_SUMMARY table here the value\_format supposed to be in JSON, you need to group all records with station id don’t  
Forget to use count aggregation function to get stations count. Sure you can test results from KSQL  
By doing ordinary SQL statements like ‘select \* from TURNSTILE\_SUMMARY;’



 RESUBMIT PROJECT

**DOWNLOAD PROJECT**

**[10](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)**[CODE REVIEW COMMENTS](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)



**Best practices for your project resubmission**

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

[Watch Video](https://review.udacity.com/?utm_campaign=ret_000_auto_ndxxx_submission-reviewed&utm_source=blueshift&utm_medium=email&utm_content=reviewsapp-submission-reviewed&bsft_clkid=7a1b5217-d0bb-49ea-9f70-82cb47382be8&bsft_uid=e53f93b0-06e4-498e-966c-3d1ea68c5365&bsft_mid=a0856cf9-1974-4992-8edc-33df624c4b2c&bsft_eid=9db03216-353a-4a26-a4a5-9c6191839f62&bsft_txnid=d061b912-ad5d-4b07-9575-685e0218e58e&bsft_mime_type=html&bsft_ek=2022-08-23T07%3A07%3A44Z&bsft_aaid=8d7e276e-4a10-41b2-8868-423fe96dd6b2&bsft_lx=3&bsft_tv=17)(3:01)