Milestone 1 Report:

Problem Statement, Data Overview, Data Acquisition & Processing, Initial Exploratory Analysis

Problem Statement:

For my second capstone, my goal was to predict campaign success on Kickstarter by leveraging NLP techniques.

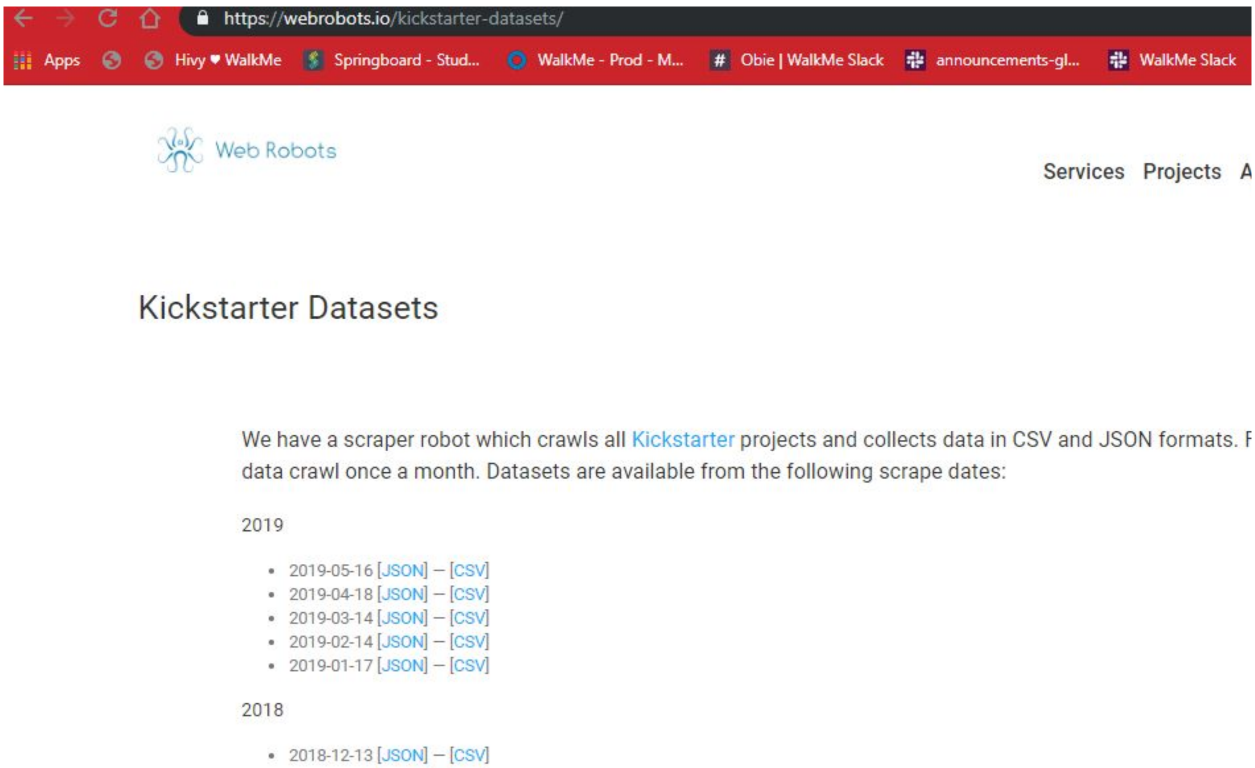
Potential parties that could be interested in this project include:

1. Potential creators wanting to understand:
    - 1) What is their realistic chance of being successful on Kickstarter
    - 2) What does the competitive landscape look like for similar categories, etc
    - 3) How they can better message their products.
2. Knock-off manufacturers wanting to understand who they should be ripping off
3. Competitive crowdfunding sites wanting to understand:
    - 1) The supply & demand of crowdfunding sites
    - 2) The products being moved (or not)

Data Overview:

Data for the project came from a series of csv files (hosted by Webrobots) which were produced by scraping the Kickstarter site once a month (https://webrobots.io/kickstarter-datasets/).

Each scrape could produce anywhere from 25-50+ csv files which were stored in folders labeled by the month and day the data was scraped. Each row in a csv file represents a single project and includes features like the creator, goal amount, country of origin and important text fields like “blurb” (a 1 sentence summary of the project), “category”, “title” and “creator”. Given the scrapes were performed once a month and the average campaign length was 30 days, the same campaign could be captured multiple times and a majority of campaigns could have been posted between scrapes.

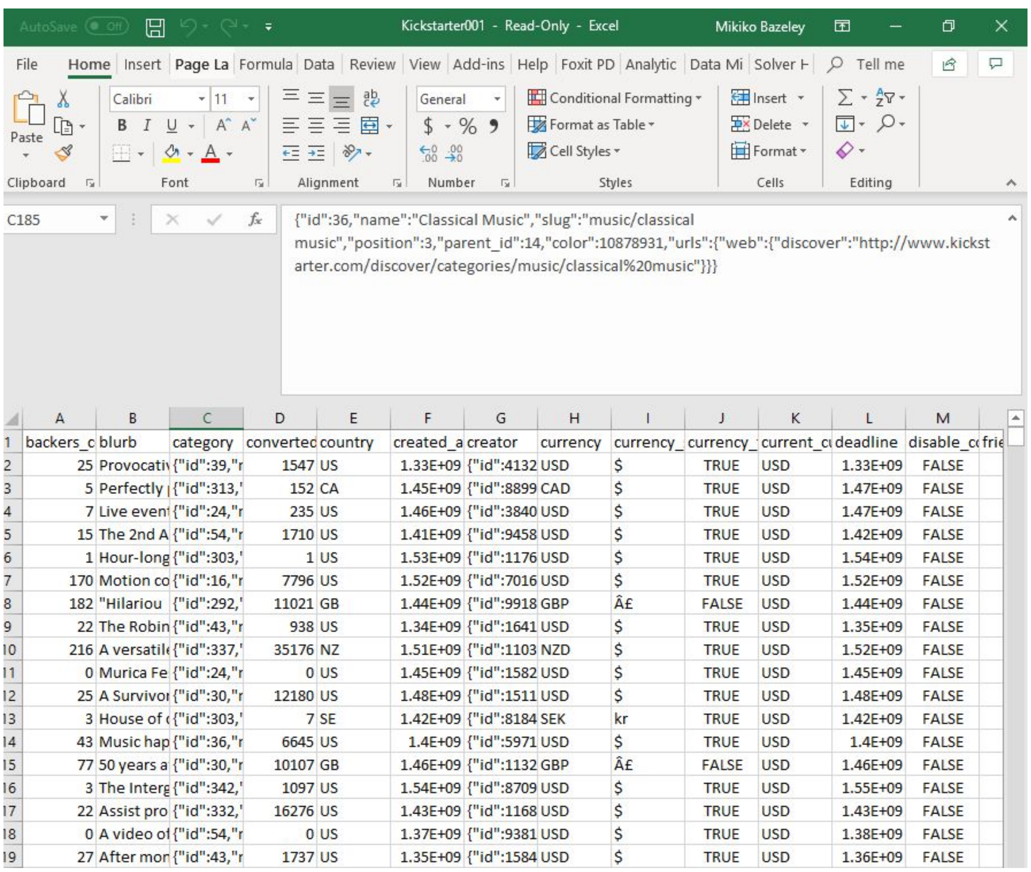


A concern of using data from a point in time too far into the campaign is producing a model that results in overly high accuracy because of signal words like “success”, “fail” or even phrases like “reached our goal ahead of schedule!”.

Given we’re trying to predict a specific outcome (“successful” or “failed”) and we have labeled data, this would be a classification problem that could be addressed using a logistic regression, random forest classifier, or similar algorithms. Our goal is to use as many of the predictors as possible, given each file contains only about 32 features. We have a mix of data types including datetime columns (deadline, state_changed_at, created_at), numeric columns (backers, goal, usd_pledged), text (blurb, title), and categorical (country, currency, state).

Deliverables will include a jupyter notebook, a summary paper, and a slide deck. The comprehensive list of possible features were:

- 'backers_count', 'blurb', 'category', 'converted_pledged_amount', 'country', 'created_at', 'creator', 'currency', 'currency_symbol', 'currency_trailing_code', 'current_currency', 'deadline', 'dirname', 'disable_communication', 'friends', 'fx_rate', 'goal', 'id', 'is_backing', 'is_starrable', 'is_starred', 'last_update_published_at', 'launched_at', 'location', 'name', 'permissions', 'photo', 'pledged', 'profile', 'slug', 'source_url', 'spotlight', 'staff_pick', 'state', 'state_changed_at', 'static_usd_rate', 'unread_messages_count', 'unseen_activity_count', 'urls', 'usd_pledged', 'usd_type'



In total the original data set was about 30.6 MB and consisted of projects scraped between Jan 28, 2016 and Feb 14, 2019. Some important considerations needed to be handled given the data collection method.

1) Projects would be duplicated between scrapes and we wanted the earliest instance as opposed to the duplicates.
2) I was only interested in using data from projects within the early days of the campaign in order to avoid data leakage.
3) Many of the columns had data that was nested in a JSON format and needed to be split out.
    a) Ex:

{"id":45,"name":"Art Books","slug":"publishing/art books","position":3,"parent_id":18,"color":14867664,"urls":{"web":{"discover":"http://www.kickstarter.com/discover/categories/publishing/art%20books"}}}

Because of the timing of the data collection, we won't be able to answer questions like:

- How did the campaigns perform over time?
- Are there differences in success factors between categories?
- Are there regional differences?
- How were edits handled within the same campaigns over the lifetime of the campaign?

Data Acquisition & Processing:

In order to begin cleaning the data I needed to first compile the 30.6MB of data spread across 100+ csv files into a single source that could be queried. I decided to leverage the sqlite3 & os python packages to create a local SQLite database that could be loaded with data from the csv files, which would allow me to define the schema, query the records, and align the common columns.

```python
def main():

    sql_delete_table = """ DROP TABLE IF EXISTS maindump;"""

    sql_create_maindump_table = """ CREATE TABLE IF NOT EXISTS maindump (
                                    ROW_ID INTEGER PRIMARY KEY AUTOINCREMENT,
                                    id integer,
                                    backers_count integer,
                                    dirname text,
                                    blurb text,
                                    category text,
                                    converted_pledged_amount integer,
                                    country text,
                                    created_at integer,
                                    creator text,
                                    currency text,
                                    currency_symbol text,
                                    currency_trailing_code text,
                                    current_currency text,
                                    deadline text,
                                    disable_communication text,
                                    friends text,
                                    fx_rate text,
                                    goal text,
                                    is_backing text,
                                    is_starrable text,
                                    is_starred text,
                                    last_update_published_at text,
                                    launched_at text,
                                    location text,
                                    name text,
                                    permissions text,
                                    photo text,
                                    pledged text,
                                    profile text,
                                    slug text,
                                    source_url text,
                                    spotlight text,
                                    staff_pick text,
                                    state text,
                                    state_changed_at text,
                                    static_usd_rate text,
                                    unread_messages_count text,
                                    unseen_activity_count text,
                                    urls text,
                                    usd_pledged text,
                                    usd_type text,
                                    scrape_date text,
                                    state_changed_at_clean text,
                                    created_at_clean text,
                                    deadline_clean text,
                                    launched_at_clean text
                                ); """
```

```python
import sqlite3
from sqlite3 import Error


def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


def create_table(conn, create_table_sql):
    """ create a table from the create_table_sql statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE statement
    :return:
    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)


def delete_table(conn, delete_table_sql):
    try:
        c = conn.cursor()
        c.execute(delete_table_sql)
    except Error as e:
        print(e)
```

```
def main():
    database = "G:\\My Drive\\sqlite\\db\\kickstarter.db"
    path = "Users\mikiko.b\Documents\Kickstarter\data"
    directory = os.path.join("c:\\",path)

    # create a database connection
    conn = create_connection(database)
    with conn:
        for dirpath, dirnames, filenames in os.walk(directory):
            global dirname
            dirname = dirpath.split(os.path.sep)[-1]
            print("dirpath:", dirpath)
            print("dirnames:", dirnames)
            print("dirname:", dirname)
            for file in filenames:
                with open(os.path.join(dirpath, file),encoding="utf8") as csvfile:
                    # remove this line
                    print("dirpath+file", os.path.join(dirpath, file))
                    reader = csv.DictReader(csvfile)
                    for row in reader:
                        a_id=row.get('id',None)
                        backers_count=row.get('backers_count',None)
                        blurb=row.get('blurb',None)
                        category=row.get('category',None)
                        converted_pledged_amount=row.get('converted_pledged_amount',None)
                        country=row.get('country',None)
                        created_at=row.get('created_at',None)
                        creator=row.get('creator',None)
                        currency=row.get('currency',None)
                        currency_symbol=row.get('currency_symbol',None)
                        currency_trailing_code=row.get('currency_trailing_code',None)
                        current_currency=row.get('current_currency',None)
                        deadline=row.get('deadline',None)
                        disable_communication=row.get('disable_communication',None)
                        friends=row.get('friends',None)
                        fx_rate=row.get('fx_rate',None)
                        goal=row.get('goal',None)
                        is_backing=row.get('is_backing',None)
                        is_starrable=row.get('is_starrable',None)
                        is_starred=row.get('is_starred',None)
                        last_update_published_at=row.get('last_update_published_at',None)
                        launched_at=row.get('launched_at',None)
                        location=row.get('location',None)
                        name=row.get('name',None)
                        permissions=row.get('permissions',None)
                        photo=row.get('photo',None)
                        pledged=row.get('pledged',None)
                        profile=row.get('profile',None)
                        slug=row.get('slug',None)
                        source_url=row.get('source_url',None)
                        spotlight=row.get('spotlight',None)
                        staff_pick=row.get('staff_pick',None)
                        state=row.get('state',None)
                        state_changed_at=row.get('state_changed_at',None)
                        static_usd_rate=row.get('static_usd_rate',None)
                        unread_messages_count=row.get('unread_messages_count',None)
                        unseen_activity_count=row.get('unseen_activity_count',None)
                        urls=row.get('urls',None)
                        usd_pledged=row.get('usd_pledged',None)
                        usd_type=row.get('usd_type',None)


                        # now I'm going to clean the dates
                        state_changed_at_clean = datetime.utcfromtimestamp(int(state_changed_at)).strftime('%Y-%m-%d %H:%M:%S
                        created_at_clean = datetime.utcfromtimestamp(int(created_at)).strftime('%Y-%m-%d %H:%M:%S')
                        deadline_clean =  datetime.utcfromtimestamp(int(deadline)).strftime('%Y-%m-%d %H:%M:%S')
                        launched_at_clean =  datetime.utcfromtimestamp(int(launched_at)).strftime('%Y-%m-%d %H:%M:%S')

                        # create scrape date
                        scrape_date = str(dirname)[12:22]

                        #combine to create an insertable row - "project"
                        #call create_project
                        campaign = (a_id,backers_count,dirname,blurb,category,converted_pledged_amount,country,created_at,cre
                        create_campaign(conn, campaign)

                print("added: ", dirname)








if __name__ == '__main__':
    main()
```

After defining the table schema, I also converted the datetime columns (state_changed_at_clean, created_at_clean, deadline_clean, launched_at_clean) from a utc timestamp to a human readable format. Once all the projects were loaded into a single table, I then needed to stage the creation of the de-duped master data set.

```
In [20]: 1 project_outcomes.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1885073 entries, 0 to 1885072
Data columns (total 10 columns):
Unnamed: 0               int64
id                       int64
slug                     object
creator                  object
created_at_clean         object
launched_at_clean        object
profile                  object
country                  object
state                    object
state_changed_at_clean   object
dtypes: int64(2), object(8)
memory usage: 143.8+ MB
```

Leveraging pandas read_sql_query function I first needed to get the list of unique projects :

projects_outcomes = pd.read_sql_query("select id, slug, creator, created_at_clean, launched_at_clean, profile, country, state, state_changed_at_clean from maindump where state != 'live';", conn)

Specifically, I selected projects that were posted within less than 15 days prior to a scrape being run. This list would consist of the projects outcomes and would have the duplicates dropped from the pandas dataframe produced from running the query.

```
In [19]: 1 project_starting.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 478 entries, 0 to 477
Data columns (total 12 columns):
Unnamed: 0           478 non-null int64
id                   478 non-null int64
slug                 478 non-null object
name                 478 non-null object
blurb                478 non-null object
category             478 non-null object
goal                 478 non-null int64
launched_at_clean    478 non-null object
deadline_clean       478 non-null object
location             478 non-null object
scrape_date          478 non-null object
state                478 non-null object
dtypes: int64(3), object(9)
memory usage: 44.9+ KB
```

The starting data, that would consist of the predictors we'd want to use to create the predictive model, was created through the following query:

projects_starting_data = pd.read_sql_query("select id, slug, name, blurb, category, goal, launched_at_clean, deadline_clean, location, scrape_date, state from maindump where (julianday(date(launched_at_clean)) - julianday(date(scrape_date))) < 15 and (julianday(date(launched_at_clean)) - julianday(date(scrape_date))) > -1;", conn)

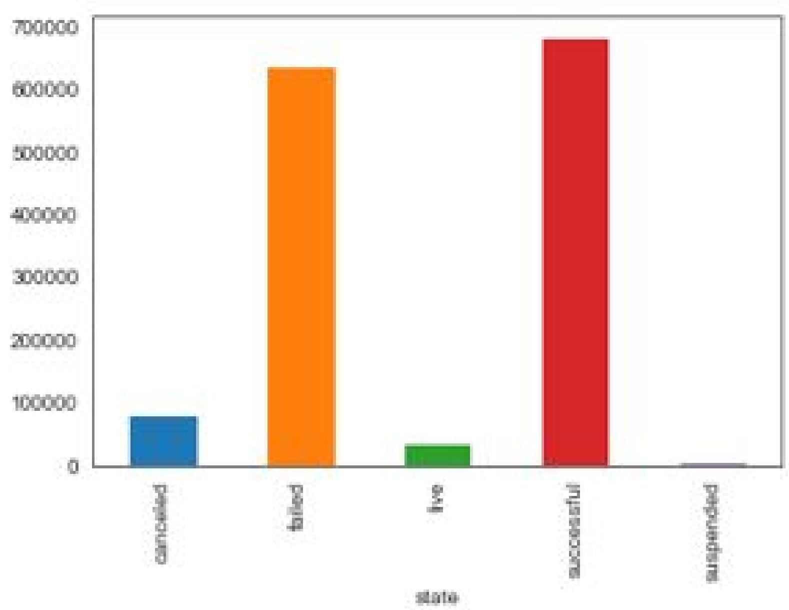
```
: 1 consolidated_table.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 478 entries, 0 to 1650
Data columns (total 21 columns):
Unnamed: 0_x             478 non-null int64
id_x                     478 non-null int64
slug                     478 non-null object
name                     478 non-null object
blurb                    478 non-null object
category                 478 non-null object
```

The starting and outcomes data sets were then merged to produce a master data set, which only included 478 campaigns.

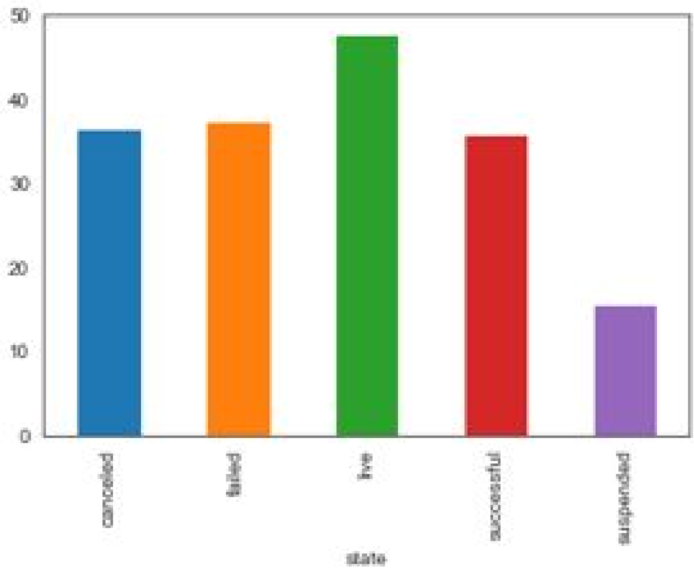
Additional cleaning including using regex to break out the nested columns, specifically “creator”, “category”, and “location” to isolate the creator name, category and country, city, state.

Initial Exploratory Analysis:

When analyzing the data I first wanted to understand if there were potential issues with class imbalance between failed and successful campaigns and found that the ratios ended up being nearly equal.



I also wanted to understand whether the average campaign duration differed by outcome and average time to launch>


50
40
30
20
10
0
canceled
failed
live
successful
suspended
state