# Intro to NLP

## #deep learning classifiers

150116884 Esra Polat
150116071 Nur Deniz Çaylı
150116028 Minel Saygısever

# Detail of Implementation

This program was implemented with Python on the Kaggle Notebook using the below libraries.

# Detail of Dataset

We used the Kickstarter Campaigns Dataset

Data on more than 20,632 Kickstarter campaigns
from Kaggle*

KICKSTARTER

```
kickstarter = pd.read_csv('../input/kickstarter-campaigns-dataset/kickstarter_data_full.csv', index_col=0)

# The dataset have 20632 entries of 67 features.
kickstarter.shape
kickstarter.info()
```

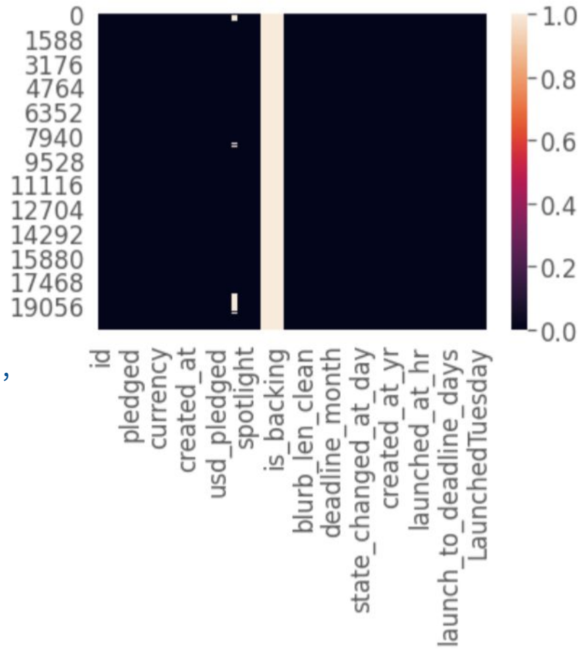
```
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          20632 non-null  int64
 1   photo                       20632 non-null  object
 2   name                        20632 non-null  object
 3   blurb                       20627 non-null  object
 4   goal                        20632 non-null  float64
 5   pledged                     20632 non-null  float64
 6   state                       20632 non-null  object
 7   slug                        20632 non-null  object
 8   disable_communication       20632 non-null  bool
 9   country                     20632 non-null  object
 10  currency                    20632 non-null  object
 11  currency_symbol             20632 non-null  object
 12  currency_trailing_code      20632 non-null  bool
 13  deadline                    20632 non-null  object
 14  state_changed_at            20632 non-null  object
 15  created_at                  20632 non-null  object
 16  launched_at                 20632 non-null  object
 17  staff_pick                  20632 non-null  bool
 18  backers_count               20632 non-null  int64
 19  static_usd_rate             20632 non-null  float64
 20  usd_pledged                 20632 non-null  float64
 21  creator                     20632 non-null  object
 22  location                    20587 non-null  object
 23  category                    18743 non-null  object
 24  profile                     20632 non-null  object
 25  spotlight                   20632 non-null  bool
 26  urls                        20632 non-null  object
 27  source_url                  20632 non-null  object
 28  friends                     60 non-null     object
 29  is_starred                  60 non-null     object
 30  is_backing                  60 non-null     object
 31  permissions                 60 non-null     object
 32  name_len                    20627 non-null  float64
 33  name_len_clean              20627 non-null  float64
 34  blurb_len                   20627 non-null  float64
 35  blurb_len_clean             20627 non-null  float64
 36  deadline_weekday            20632 non-null  object
 37  state_changed_at_weekday    20632 non-null  object
 38  created_at_weekday          20632 non-null  object
 39  launched_at_weekday         20632 non-null  object
 40  deadline_month              20632 non-null  int64
 41  deadline_day                20632 non-null  int64
 42  deadline_yr                 20632 non-null  int64
 43  deadline_hr                 20632 non-null  int64
 44  state_changed_at_month      20632 non-null  int64
 45  state_changed_at_day        20632 non-null  int64
 46  state_changed_at_yr         20632 non-null  int64
 47  state_changed_at_hr         20632 non-null  int64
 48  created_at_month            20632 non-null  int64
 49  created_at_day              20632 non-null  int64
 50  created_at_yr               20632 non-null  int64
 51  created_at_hr               20632 non-null  int64
 52  launched_at_month           20632 non-null  int64
 53  launched_at_day             20632 non-null  int64
 54  launched_at_yr              20632 non-null  int64
 55  launched_at_hr              20632 non-null  int64
 56  create_to_launch            20632 non-null  object
 57  launch_to_deadline          20632 non-null  object
 58  launch_to_state_change      20632 non-null  object
 59  create_to_launch_days       20632 non-null  int64
 60  launch_to_deadline_days     20632 non-null  int64
 61  launch_to_state_change_days 20632 non-null  int64
 62  SuccessfulBool              20632 non-null  int64
 63  USorGB                      20632 non-null  int64
 64  TOPCOUNTRY                  20632 non-null  int64
 65  LaunchedTuesday             20632 non-null  int64
 66  DeadlineWeekend             20632 non-null  int64
dtypes: bool(4), float64(8), int64(26), object(29)
memory usage: 10.2+ MB
```

# View Dataset

```
# looking here we can see that 'is_backing' and 'profile' contain missing values
sns.heatmap(kickstarter.isnull())

# friends, is_starred, is_backing, and permissions are looking weird
kickstarter['friends'].isnull().value_counts()
cols_to_drop = ['friends', 'is_starred', 'is_backing', 'permissions']
kickstarter.drop(labels=cols_to_drop, axis=1, inplace=True)
kickstarter.drop(labels='profile', axis=1, inplace=True)

#there are a lot of unnecessary features
second_col_drop = ['id','photo','slug','currency_symbol','currency_trailing_code',
'creator','location', 'urls','source_url','name_len','blurb_len','create_to_launch',
'launch_to_deadline', 'launch_to_state_change','USorGB','TOPCOUNTRY','LaunchedTuesday',
'DeadlineWeekend', 'deadline_month','deadline_day','deadline_yr','deadline_hr',
'state_changed_at_month','state_changed_at_day','state_changed_at_yr','state_changed_at_hr',
'created_at_month','created_at_day','created_at_yr','created_at_hr','launched_at_month',
'launched_at_day', 'launched_at_yr','launched_at_hr']

kickstarter.drop(labels=second_col_drop, axis=1, inplace=True)

# we reduced dimensionality from 67 to 28
kickstarter.shape
```

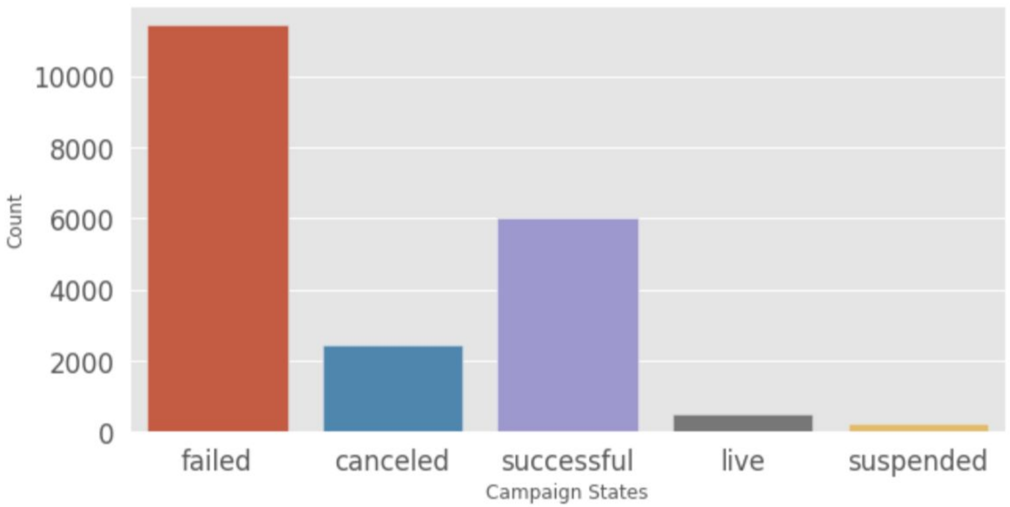
```
(20632, 28)
```

```
#converts type bool to 0 for false and 1 for true
kickstarter['disable_communication'] = kickstarter['disable_communication'] * 1
kickstarter['staff_pick'] = kickstarter['staff_pick'] * 1
kickstarter['spotlight'] = kickstarter['spotlight'] * 1
```

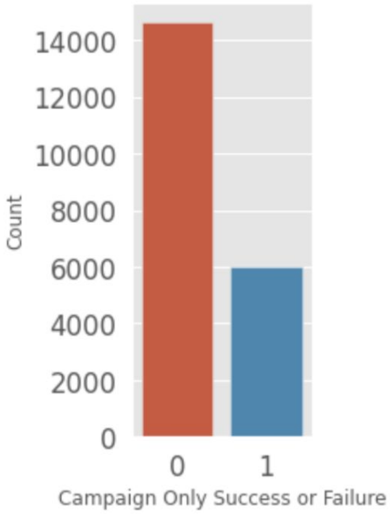


# View Dataset

```
figsize(10, 5)
sns.countplot(kickstarter['state']);
plt.xlabel('Campaign States');
plt.ylabel('Count');
```




```
figsize(2, 5)
sns.countplot(kickstarter['SuccessfulBool']);
plt.xlabel('Campaign Only Success or Failure');
plt.ylabel('Count');
```




Only 29.17% of campaigns were successful.

# Interpretation

```
# when we look at the general statistics, we see how each feature covers a very different range.
kickstarter.describe().transpose()
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **goal** | 20632.0 | 94104.965285 | 1.335511e+06 | 1.000000 | 4000.0 | 14000.000000 | 50000.000000 | 1.000000e+08 |
| **pledged** | 20632.0 | 21392.675739 | 1.204973e+05 | 0.000000 | 25.0 | 695.000000 | 5954.250000 | 6.225355e+06 |
| **disable_communication** | 20632.0 | 0.011148 | 1.049952e-01 | 0.000000 | 0.0 | 0.000000 | 0.000000 | 1.000000e+00 |
| **staff_pick** | 20632.0 | 0.105903 | 3.077215e-01 | 0.000000 | 0.0 | 0.000000 | 0.000000 | 1.000000e+00 |
| **backers_count** | 20632.0 | 183.675843 | 1.222013e+03 | 0.000000 | 2.0 | 12.000000 | 63.000000 | 1.058570e+05 |
| **static_usd_rate** | 20632.0 | 1.039363 | 2.304189e-01 | 0.045641 | 1.0 | 1.000000 | 1.000000 | 1.715913e+00 |
| **usd_pledged** | 20632.0 | 20915.907911 | 1.154717e+05 | 0.000000 | 25.0 | 716.301193 | 6004.628177 | 6.225355e+06 |
| **spotlight** | 20632.0 | 0.291683 | 4.545481e-01 | 0.000000 | 0.0 | 0.000000 | 1.000000 | 1.000000e+00 |
| **name_len_clean** | 20627.0 | 5.292578 | 2.418168e+00 | 1.000000 | 3.0 | 5.000000 | 7.000000 | 1.400000e+01 |
| **blurb_len_clean** | 20627.0 | 13.081204 | 3.283547e+00 | 1.000000 | 11.0 | 13.000000 | 15.000000 | 3.000000e+01 |
| **create_to_launch_days** | 20632.0 | 49.577598 | 1.110946e+02 | 0.000000 | 3.0 | 14.000000 | 45.000000 | 1.754000e+03 |
| **launch_to_deadline_days** | 20632.0 | 34.716896 | 1.187314e+01 | 1.000000 | 30.0 | 30.000000 | 40.000000 | 9.100000e+01 |
| **launch_to_state_change_days** | 20632.0 | 31.169397 | 1.427971e+01 | 0.000000 | 28.0 | 30.000000 | 35.000000 | 9.100000e+01 |
| **SuccessfulBool** | 20632.0 | 0.291683 | 4.545481e-01 | 0.000000 | 0.0 | 0.000000 | 1.000000 | 1.000000e+00 |

# Interpretation

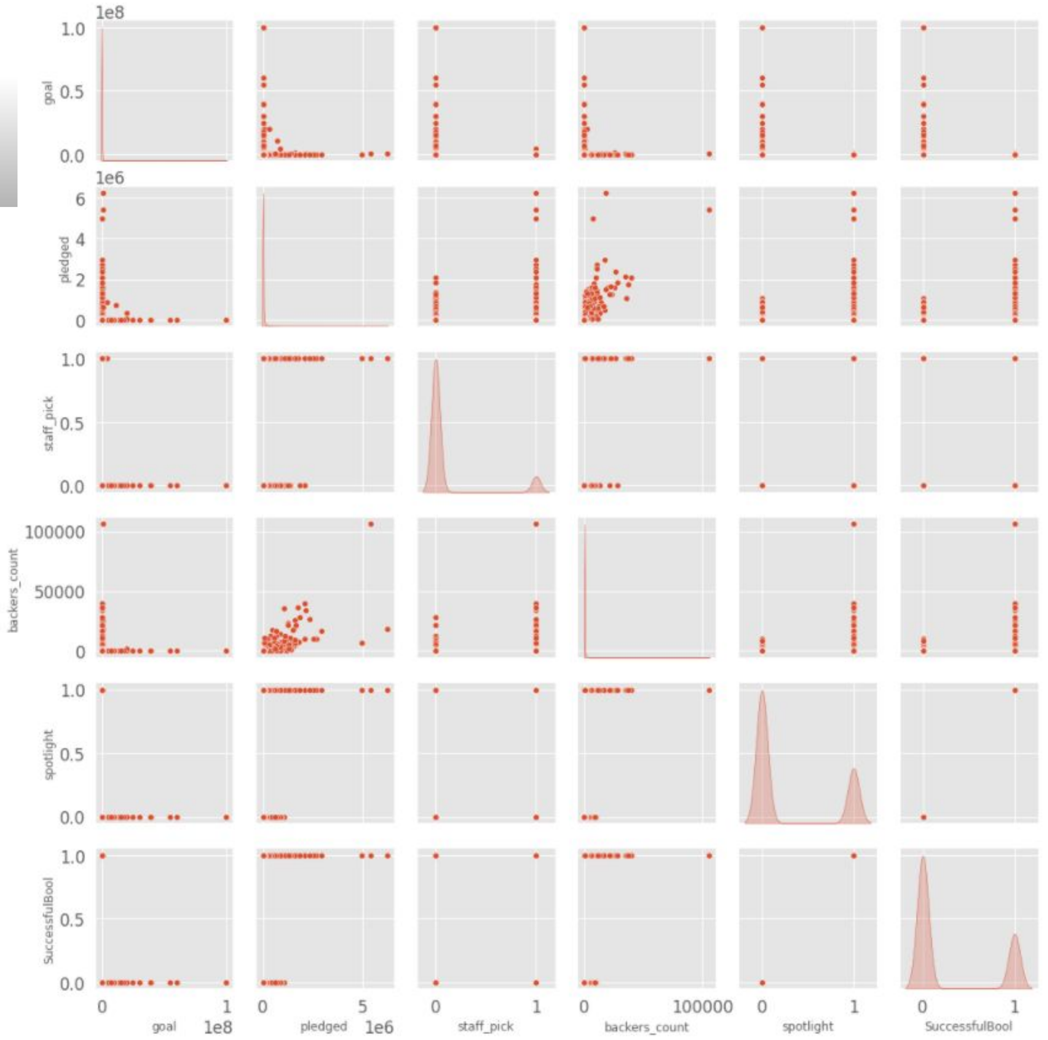
```
# Let's take a quick look at the common distribution of a
few pairs of columns
sns.pairplot(kickstarter[['goal','pledged','staff_pick',
'backers_count', 'spotlight','SuccessfulBool']],
diag_kind='kde')
```

```
# It looks like the goal variable has a huge spread
kickstarter['goal'].sort_values().tail()
```

```
3487      40000000.0
11043     55000000.0
8678      60000000.0
4801     100000000.0
8696     100000000.0
Name: goal, dtype: float64
```

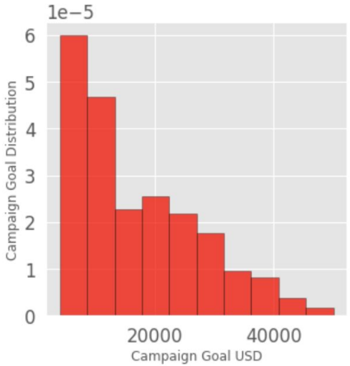
```
# the pledged amount is more reasonable because this
represents real money that people decided to give
kickstarter['pledged'].sort_values().tail()
```

```
8763      2708472.39
12829     2952508.59
12911     4961032.74
4363      5408916.95
8805      6225354.98
Name: pledged, dtype: float64
```



# Interpretation

```
first_quartile = kickstarter['goal'].describe()['25%']
third_quartile = kickstarter['goal'].describe()['75%']
iqr = third_quartile - first_quartile
kickstarter_goal_iqr = kickstarter[(kickstarter['goal'] > first_quartile) & (kickstarter['goal'] < third_quartile)]

kickstarter_goal_iqr.describe().transpose()

figsize(5, 5)
plt.hist(kickstarter_goal_iqr['goal'], 10, density = 10,
        color ='red',edgecolor = 'black', alpha = 0.7)
plt.xlabel('Campaign Goal USD')
plt.ylabel('Campaign Goal Distribution')
plt.show()
```




| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **goal** | 10107.0 | 16841.510237 | 10887.732017 | 4059.000000 | 8000.0 | 15000.0 | 25000.00 | 4.999900e+04 |
| **pledged** | 10107.0 | 14891.312073 | 67578.044235 | 0.000000 | 28.0 | 840.0 | 8489.00 | 2.344135e+06 |
| **disable_communication** | 10107.0 | 0.011477 | 0.106520 | 0.000000 | 0.0 | 0.0 | 0.00 | 1.000000e+00 |
| **staff_pick** | 10107.0 | 0.109330 | 0.312068 | 0.000000 | 0.0 | 0.0 | 0.00 | 1.000000e+00 |
| **backers_count** | 10107.0 | 173.063125 | 907.532746 | 0.000000 | 2.0 | 13.0 | 82.00 | 3.678100e+04 |
| **static_usd_rate** | 10107.0 | 1.036482 | 0.202281 | 0.045704 | 1.0 | 1.0 | 1.00 | 1.715913e+00 |
| **usd_pledged** | 10107.0 | 15099.692854 | 67484.061872 | 0.000000 | 29.0 | 850.0 | 8618.75 | 2.344135e+06 |
| **spotlight** | 10107.0 | 0.266053 | 0.441914 | 0.000000 | 0.0 | 0.0 | 1.00 | 1.000000e+00 |
| **name_len_clean** | 10105.0 | 5.366452 | 2.431118 | 1.000000 | 3.0 | 6.0 | 7.00 | 1.400000e+01 |
| **blurb_len_clean** | 10105.0 | 13.073726 | 3.285475 | 1.000000 | 11.0 | 13.0 | 15.00 | 3.000000e+01 |
| **create_to_launch_days** | 10107.0 | 53.305036 | 113.766666 | 0.000000 | 4.0 | 16.0 | 50.00 | 1.692000e+03 |
| **launch_to_deadline_days** | 10107.0 | 34.896309 | 11.219802 | 1.000000 | 30.0 | 30.0 | 40.00 | 9.100000e+01 |
| **launch_to_state_change_days** | 10107.0 | 31.493915 | 13.800788 | 0.000000 | 29.0 | 30.0 | 35.00 | 9.100000e+01 |
| **SuccessfulBool** | 10107.0 | 0.266053 | 0.441914 | 0.000000 | 0.0 | 0.0 | 1.00 | 1.000000e+00 |

# Outlier Data

```
# trim backers_count, pledged and create_to_launch_days then create
# a new IQR dataframe with these truncated values
kickstarter_iqr_trimmed = kickstarter_goal_iqr

first_quartile = kickstarter['create_to_launch_days'].describe()['25%']
third_quartile = kickstarter['create_to_launch_days'].describe()['75%']
iqr = third_quartile - first_quartile
kickstarter_iqr_trimmed = kickstarter[(kickstarter['create_to_launch_days'] >
      first_quartile) & (kickstarter['create_to_launch_days'] < third_quartile)]

first_quartile = kickstarter['pledged'].describe()['25%']
third_quartile = kickstarter['pledged'].describe()['75%']
iqr = third_quartile - first_quartile
kickstarter_iqr_trimmed = kickstarter[(kickstarter['pledged'] >
      first_quartile) & (kickstarter['pledged'] < third_quartile)]

first_quartile = kickstarter['backers_count'].describe()['25%']
third_quartile = kickstarter['backers_count'].describe()['75%']
iqr = third_quartile - first_quartile
kickstarter_iqr_trimmed = kickstarter[(kickstarter['backers_count'] >
      first_quartile) & (kickstarter['backers_count'] < third_quartile)]

# This reduction resulted in a dataframe where there are 9308 instances,
#  with only the IQR for the variables in question remaining.
len(kickstarter_iqr_trimmed)
```

```
9308
```

```
# correlations btw each variable against SuccessfulBool, which
remember, is a binary value where 0=failed and 1=succeeded
kickstarter_iqr_trimmed.corr()['SuccessfulBool'].sort_values()
```

```
launch_to_deadline_days       -0.184938
create_to_launch_days         -0.085983
disable_communication         -0.053091
launch_to_state_change_days   -0.047071
goal                          -0.033484
name_len_clean                -0.027807
blurb_len_clean                0.058825
staff_pick                     0.109232
static_usd_rate                0.109604
pledged                        0.133948
usd_pledged                    0.181088
backers_count                  0.404936
spotlight                      1.000000
SuccessfulBool                 1.000000
Name: SuccessfulBool, dtype: float64
```

# Outlier Data

```
# Looking at the correlations above we can see that nothing is too strongly correlated except spotlight, backers_count, pledged, and staff_pick
# But really the only significant ones are backers_count and spotlight
len(kickstarter_iqr_trimmed[kickstarter_iqr_trimmed['spotlight'] == 1])
```

```
2200
```

```
# taken together with the spotlight variable's correlation to SuccessfulBool, we can conclude that all spotlighted campaigns were successful,
# at least in this dataset, taking into account the fact that it is reduced to IQR values only
len(kickstarter_iqr_trimmed[kickstarter_iqr_trimmed['SuccessfulBool'] == 1])
```

```
2200
```

```
# we are going to pool together these strongly correlated features for feature selection
reduced_x_features = kickstarter_iqr_trimmed[['launch_to_deadline_days', 'staff_pick', 'pledged', 'backers_count', 'spotlight', 'goal']]
reduced_y = kickstarter_iqr_trimmed[['SuccessfulBool']]

# Because of the original format of the variables, we need to take the log and sqrt transformations of them and  check correlation with those
# as well to account for non-linear relationships
numeric_subset = kickstarter_iqr_trimmed.select_dtypes('number')

for col in numeric_subset.columns:
    if col == 'SuccessfulBool': next
    else: numeric_subset['sqrt_' + col] = np.sqrt(numeric_subset[col])
          numeric_subset['log_' + col] = np.log(numeric_subset[col])

categorical_subset = kickstarter_iqr_trimmed['category']
categorical_subset = pd.get_dummies(categorical_subset)
features = pd.concat([numeric_subset, categorical_subset], axis = 1)
features = features.dropna(subset = ['SuccessfulBool'])
correlations = features.corr()['SuccessfulBool'].dropna().sort_values()
correlations.head()
```

```
log_goal                        -0.554957
sqrt_goal                       -0.272015
log_launch_to_deadline_days     -0.219717
sqrt_launch_to_deadline_days    -0.205036
launch_to_deadline_days         -0.184938
Name: SuccessfulBool, dtype: float64
```

# Outlier Data

```
# we saw in the previous step that goal got a boost in correlation what you take its log, so we will add log_goal into the reduced_x_features
# dataframe and saw log_pledged show a significant boost as well, so that will be included

reduced_x_features['log_goal'] = features['log_goal']
reduced_x_features['log_pledged'] = features['log_pledged']
```

```
reduced_x_features
```

| | launch_to_deadline_days | staff_pick | pledged | backers_count | spotlight | goal | log_goal | log_pledged |
|---|---|---|---|---|---|---|---|---|
| **2** | 60 | 0 | 120.0 | 5 | 0 | 100000.0 | 11.512925 | 4.787492 |
| **4** | 32 | 0 | 356.0 | 17 | 0 | 3222.0 | 8.077758 | 5.874931 |
| **5** | 30 | 0 | 1136.0 | 12 | 0 | 13000.0 | 9.472705 | 7.035269 |
| **8** | 30 | 0 | 153.0 | 7 | 0 | 6000.0 | 8.699515 | 5.030438 |
| **10** | 30 | 0 | 72.0 | 5 | 0 | 7300.0 | 8.895630 | 4.276666 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **20624** | 30 | 0 | 761.0 | 4 | 0 | 40000.0 | 10.596635 | 6.634633 |
| **20625** | 60 | 0 | 3075.0 | 34 | 0 | 20000.0 | 9.903488 | 8.031060 |
| **20626** | 35 | 0 | 101.0 | 9 | 0 | 5000.0 | 8.517193 | 4.615121 |
| **20628** | 30 | 0 | 1559.0 | 13 | 0 | 100000.0 | 11.512925 | 7.351800 |
| **20631** | 30 | 0 | 380.0 | 10 | 0 | 50000.0 | 10.819778 | 5.940171 |

9308 rows × 8 columns

```
reduced_y
```

| | SuccessfulBool |
|---|---|
| **2** | 0 |
| **4** | 0 |
| **5** | 0 |
| **8** | 0 |
| **10** | 0 |
| **...** | ... |
| **20624** | 0 |
| **20625** | 0 |
| **20626** | 0 |
| **20628** | 0 |
| **20631** | 0 |

9308 rows × 1 columns

# View Dataset

```
# when we transformed goal and pledged to log_goal and log_pledged, we found that these had a stronger correlation than their original forms,
# so these new features were added to reduced_x_feature
figsize(14,5)
sns.heatmap(kickstarter_iqr_trimmed.corr(), annot=True, annot_kws={"size": 9}, cmap="Purples")
```

# Before the model..

```
kickstarter_X = []
kickstarter_y = []
for i, j in reduced_x_features.iterrows():
    tmp = str(reduced_x_features['launch_to_deadline_days'][i]) + " " + str(reduced_x_features['staff_pick'][i]) + " " + \
        str(reduced_x_features['backers_count'][i]) + " " + str(reduced_x_features['spotlight'][i]) + " " + \
        str(reduced_x_features['goal'][i]) + " " + str(reduced_x_features['log_goal'][i]) + " " + str(reduced_x_features['log_pledged'][i])
    kickstarter_X.append(tmp)
    kickstarter_y.append(int(reduced_y['SuccessfulBool'][i]))


max_words = 2000
max_length = 30
vector_length = 16

encoded_docs = [one_hot(d, max_words) for d in kickstarter_X]
padded_docs = pad_sequences(encoded_docs, maxlen=6, padding='post')

X_train, X_test, y_train, y_test = train_test_split(padded_docs,
                                                    np.array(kickstarter_y)[:, None].astype(int),
                                                    test_size=0.20,
                                                    random_state=1234)
```

kickstarter_X

```
['60 0 5 0 100000.0 11.512925464970229 4.787491742782046',
 '32 0 17 0 3222.0 8.07775756373692 5.87493073085203',
 '30 0 12 0 13000.0 9.472704636443673 7.035268599281097',
 '30 0 7 0 6000.0 8.699514748210191 5.030437921392435',
 '30 0 5 0 7300.0 8.895629627136483 4.276666119016055',
 '60 0 10 0 10000.0 9.210340371976184 6.716594773520978',
 '45 0 7 0 10000.0 9.210340371976184 5.940171252720432',
 '60 0 3 0 2000.0 7.600902459542082 4.0943445622221',
 '20 0 11 0 2000.0 7.600902459542082 5.545177444479562',
 '20 0 18 0 5275.0 8.570733958344267 7.399398083331354',
 '30 0 4 0 1280.0 7.154615356913663 3.044522437723423',
 '30 0 4 0 20000.0 9.903487552536127 4.110873864173311',
 '28 0 35 0 4500.0 8.411832675758411 6.536691597591305',
 '30 0 6 0 450.0 6.1092475827643655 3.4339872044851463',
 '30 0 4 0 9000.0 9.104979856318357 3.784189633918261',
 '32 0 10 0 1200.0 7.090076835776092 6.1675164908883415',
 '40 0 14 0 1000.0 6.907755278982137 4.060443010546419',
 '30 0 3 0 1500.0 7.313220387090301 4.787491742782046',
 '30 0 3 0 2988.0 8.002359546252707 5.488937726156687',
 '30 0 6 0 1500.0 7.313220387090301 4.499809670330265',
 '14 0 4 0 2500.0 7.824046010856292 4.465908118654584',
 '50 0 7 0 10000.0 9.210340371976184 3.5553480614894135',
 '30 0 21 0 25000.0 10.126631103850338 7.531552381407289',
 '35 0 5 0 19778.0 9.892325487829936 3.7376696182833684',
 '31 0 4 0 15350.0 9.638870753015343 5.247024072160486',
 '37 0 8 0 28500.0 10.257659366256743 6.620073206530356',
 '60 0 8 0 1000.0 6.907755278982137 5.081404364984463',
```

kickstarter_y

```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
```

# Data Sets

```
X_train
array([[  86, 1124,  142,  410,  369, 1143],
       [ 481, 1124,  369, 1669,  369, 1015],
       [ 598, 1124, 1134, 1862, 1038, 1914],
       ...,
       [1579, 1124,  370, 1946,  369,  833],
       [1430, 1124,  217, 1970,  217,  167],
       [1134, 1124, 1038,  673,  799,   48]], dtype=int32)
```

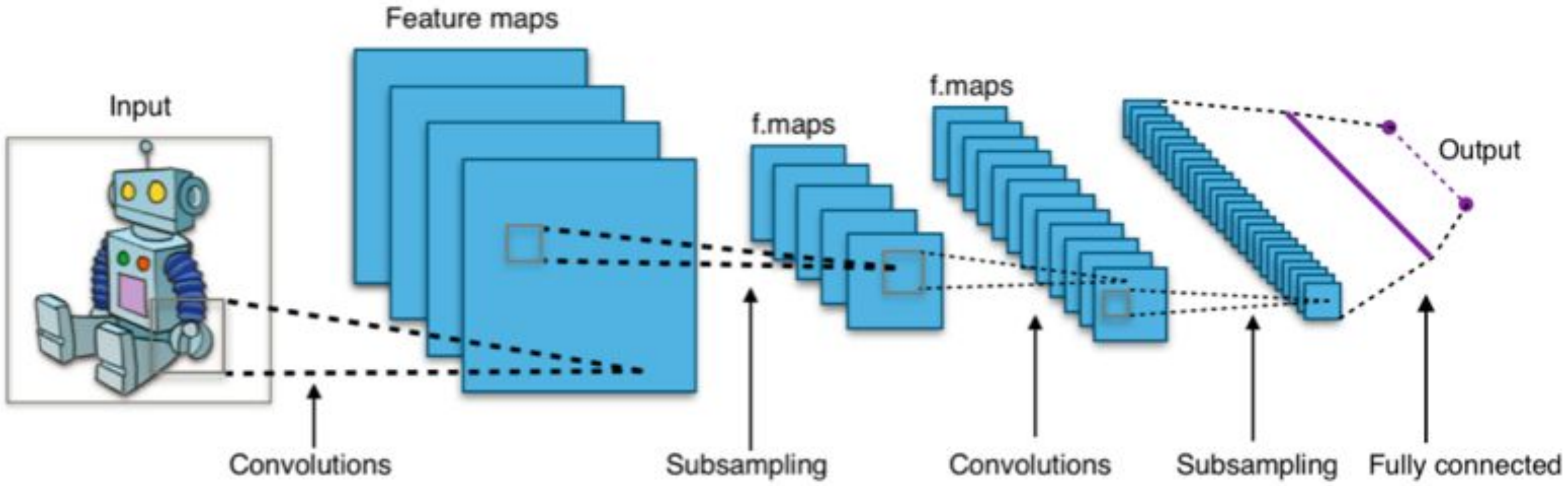
```
y_train
array([[0],
       [1],
       [0],
       ...,
       [0],
       [1],
       [1]])
```

```
X_test
array([[1802, 1124, 1134, 1005,  757, 1382],
       [ 834, 1124,   16, 1375,  217, 1997],
       [ 214, 1124,  217,  249, 1189,  806],
       ...,
       [ 763, 1124,  295, 1395,  295,  632],
       [ 481, 1124,  369, 1669,  217,  627],
       [  73, 1124, 1134, 1645,  295, 1714]], dtype=int32)
```

```
y_test
array([[0],
       [0],
       [0],
       ...,
       [1],
       [0],
       [0]])
```

# CNN (Convolutional Neural Network)

Convolutional neural network (CNN, or ConvNet) is a class of deep neural network, most commonly applied to analyze visual imagery.[1] They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps.[2][3]



# Prediction of Successful with CNN

```
# Initialising the RNN
model = Sequential()
# Adding the first CNN layer and Dropout layer
model.add(Dense(128, activation="relu", input_shape=(X_train.shape[1],)))
model.add(Dropout(0.2))
# Adding a second CNN layer and Dropout layer
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.2))
# Adding a third CNN layer and Dropout layer
model.add(Dense(32, activation="relu"))
model.add(Dropout(0.2))
# Adding a fourth CNN layer and Dropout layer
model.add(Dense(16, activation="relu"))
model.add(Dropout(0.2))
# For Full connection layer we use dense as the output is 1D so we use unit=1 adding the output layer
model.add(Dense(1))

print(model.summary())
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['acc'])

history = model.fit(X_train, y_train, epochs=50, verbose=1,validation_data=(X_test, y_test), batch_size=256)
scores = model.evaluate(X_test, y_test, verbose=1,batch_size = 256)

dt = RandomForestRegressor(criterion='mae', n_jobs=-1, n_estimators=10,max_depth=7, min_samples_leaf=1, random_state=3)
dt.fit(X_train,y_train)
y_predicted = dt.predict(X_test)
accuracy = dt.score(X_test,y_test)
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 30, 16)            32016
_________________________________________________________________
dense_30 (Dense)             (None, 30, 128)           2176
_________________________________________________________________
dropout_24 (Dropout)         (None, 30, 128)           0
_________________________________________________________________
dense_31 (Dense)             (None, 30, 64)            8256
_________________________________________________________________
dropout_25 (Dropout)         (None, 30, 64)            0
_________________________________________________________________
dense_32 (Dense)             (None, 30, 32)            2080
_________________________________________________________________
dropout_26 (Dropout)         (None, 30, 32)            0
_________________________________________________________________
dense_33 (Dense)             (None, 30, 16)            528
_________________________________________________________________
dropout_27 (Dropout)         (None, 30, 16)            0
_________________________________________________________________
dense_34 (Dense)             (None, 30, 1)             17
=================================================================
Total params: 45,073
Trainable params: 45,073
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
30/30 [==============================] - 2s 24ms/step - loss: 0.7131 - accuracy:
Epoch 2/50
30/30 [==============================] - 0s 14ms/step - loss: 0.5336 - accuracy:
Epoch 3/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4996 - accuracy:
Epoch 4/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4766 - accuracy:
Epoch 5/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4694 - accuracy:
Epoch 6/50
30/30 [==============================] - 0s 15ms/step - loss: 0.4786 - accuracy:
Epoch 7/50
30/30 [==============================] - 0s 16ms/step - loss: 0.4679 - accuracy:
Epoch 8/50
30/30 [==============================] - 0s 15ms/step - loss: 0.4674 - accuracy:
```

```
30/30 [==============================] - 0s 14ms/step - loss: 0.4741 - accuracy: 0.7633 - val_loss: 0.5071 - val_accuracy: 0.7712
Epoch 38/50
30/30 [==============================] - 0s 15ms/step - loss: 0.4615 - accuracy: 0.7745 - val_loss: 0.5158 - val_accuracy: 0.7726
Epoch 39/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4577 - accuracy: 0.7768 - val_loss: 0.5117 - val_accuracy: 0.7718
Epoch 40/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4579 - accuracy: 0.7774 - val_loss: 0.5161 - val_accuracy: 0.7726
Epoch 41/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4583 - accuracy: 0.7750 - val_loss: 0.5032 - val_accuracy: 0.7715
Epoch 42/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4504 - accuracy: 0.7845 - val_loss: 0.5170 - val_accuracy: 0.7697
Epoch 43/50
30/30 [==============================] - 0s 15ms/step - loss: 0.4638 - accuracy: 0.7691 - val_loss: 0.5037 - val_accuracy: 0.7697
Epoch 44/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4626 - accuracy: 0.7697 - val_loss: 0.5149 - val_accuracy: 0.7714
Epoch 45/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4549 - accuracy: 0.7771 - val_loss: 0.5148 - val_accuracy: 0.7712
Epoch 46/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4552 - accuracy: 0.7781 - val_loss: 0.5100 - val_accuracy: 0.7697
Epoch 47/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4590 - accuracy: 0.7728 - val_loss: 0.5134 - val_accuracy: 0.7726
Epoch 48/50
30/30 [==============================] - 0s 16ms/step - loss: 0.4619 - accuracy: 0.7733 - val_loss: 0.5117 - val_accuracy: 0.7709
Epoch 49/50
30/30 [==============================] - 0s 15ms/step - loss: 0.4535 - accuracy: 0.7788 - val_loss: 0.5144 - val_accuracy: 0.7723
Epoch 50/50
30/30 [==============================] - 0s 14ms/step - loss: 0.4583 - accuracy: 0.7740 - val_loss: 0.5026 - val_accuracy: 0.7691
233/233 [==============================] - 0s 2ms/step - loss: 0.4513 - accuracy: 0.7757
Training Accuracy: 0.7757
59/59 [==============================] - 0s 2ms/step - loss: 0.5026 - accuracy: 0.7691
Testing Accuracy:  0.7691
8/8 [==============================] - 0s 4ms/step - loss: 0.5026 - accuracy: 0.7691
Accuracy: 76.91%
Training Accuracy: 0.4108917925899547
Testing Accuracy: 0.3916973302822272
Mean Squared Error 0.10925886143931257
Testing Accuracy: 0.3916973302822272
```

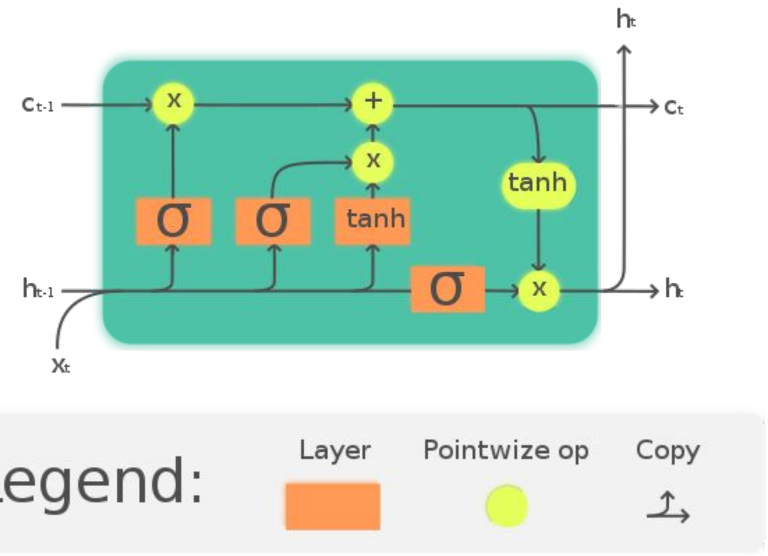
```
Accuracy: 76.91%
Training Accuracy: 0.4108917925899547
Testing Accuracy: 0.3916973302822272
Mean Squared Error 0.10925886143931257
Testing Accuracy: 0.3916973302822272
```

a few
example
from results

| X | y (actual) | Predicted | |
|---|---|---|---|
| [1955 1551 1753 1040 1753 1398] | [1] | 0.0 | ✖ |
| [ 330 1551 1922 1218 1922 1747] | [1] | 1.0 | ✔ |
| [ 950 1551 1922 81 336 1307] | [1] | 1.0 | ✔ |
| [ 950 1551 1922 81 336 801] | [1] | 1.0 | ✔ |
| [1978 1551 1749 540 1922 1359] | [0] | 0.0 | ✔ |
| [1451 1551 1922 1530 1922 1879] | [0] | 1.0 | ✖ |
| [ 950 1551 1922 81 1922 1555] | [1] | 1.0 | ✔ |
| [ 950 1551 1922 81 1922 1639] | [0] | 1.0 | ✖ |
| [1451 1551 1922 1530 1922 423] | [1] | 1.0 | ✔ |

# LSTM (Long Short-Term Memory)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time.



# Prediction of Successful with LSTM

```
# Initialising the RNN
model = Sequential()
model.add(layers.Embedding(max_words+1, vector_length, input_length=max_length))
# Adding the first LSTM layer and Dropout layer
model.add(LSTM(units = 128, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
# Adding a second LSTM layer and Dropout layer
model.add(LSTM(units = 64, return_sequences = True))
model.add(Dropout(0.2))
# Adding a third LSTM layer and Dropout layer
model.add(LSTM(units = 32, return_sequences = True))
model.add(Dropout(0.2))
# Adding a fourth LSTM layer and Dropout layer
model.add(LSTM(units = 16))
model.add(Dropout(0.2))
# For full connection layer we use dense as the output is 1D so we use unit=1 adding the output layer
model.add(Dense(1, activation= 'relu'))

print(model.summary())
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=50, verbose=1,validation_data=(X_test, y_test),batch_size=256)
scores = model.evaluate(X_test, y_test, verbose=1,batch_size = 256)

dt = RandomForestRegressor(criterion='mae',n_jobs=-1, n_estimators=10,max_depth=6, min_samples_leaf=1, random_state=3)
dt.fit(X_train,y_train)
y_predicted = dt.predict(X_test)
accuracy = dt.score(X_test,y_test)
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 30, 16)            32016
_________________________________________________________________
lstm (LSTM)                  (None, 30, 128)           74240
_________________________________________________________________
dropout_28 (Dropout)         (None, 30, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 64)            49408
_________________________________________________________________
dropout_29 (Dropout)         (None, 30, 64)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 30, 32)            12416
_________________________________________________________________
dropout_30 (Dropout)         (None, 30, 32)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 16)                3136
_________________________________________________________________
dropout_31 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_35 (Dense)             (None, 1)                 17
=================================================================
Total params: 171,233
Trainable params: 171,233
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
30/30 [==============================] - 10s 110ms/step - loss: 1.0772 - accuracy: 0.7646 - v
Epoch 2/50
30/30 [==============================] - 2s 60ms/step - loss: 0.4980 - accuracy: 0.7711 - va
Epoch 3/50
30/30 [==============================] - 2s 59ms/step - loss: 0.4066 - accuracy: 0.8027 - va
Epoch 4/50
30/30 [==============================] - 2s 59ms/step - loss: 0.4055 - accuracy: 0.8095 - va
Epoch 5/50
30/30 [==============================] - 2s 60ms/step - loss: 0.3730 - accuracy: 0.8203 - va
Epoch 6/50
30/30 [==============================] - 2s 66ms/step - loss: 0.3644 - accuracy: 0.8340 - va
Epoch 7/50
30/30 [==============================] - 2s 62ms/step - loss: 0.3692 - accuracy: 0.8307 - va
Epoch 8/50
30/30 [==============================] - 2s 59ms/step - loss: 0.3670 - accuracy: 0.8383 - va
Epoch 9/50
30/30 [==============================] - 2s 60ms/step - loss: 0.3691 - accuracy: 0.8499 - va
Epoch 10/50
```

```
30/30 [==============================] - 2s 59ms/step - loss: 0.1374 - accuracy: 0.9708 - val_loss: 0.5616 - val_accuracy: 0.8937
Epoch 34/50
30/30 [==============================] - 2s 59ms/step - loss: 0.1587 - accuracy: 0.9615 - val_loss: 0.6407 - val_accuracy: 0.8904
Epoch 35/50
30/30 [==============================] - 2s 60ms/step - loss: 0.1707 - accuracy: 0.9419 - val_loss: 0.6336 - val_accuracy: 0.8899
Epoch 36/50
30/30 [==============================] - 2s 60ms/step - loss: 0.1475 - accuracy: 0.9663 - val_loss: 0.5270 - val_accuracy: 0.8883
Epoch 37/50
30/30 [==============================] - 2s 60ms/step - loss: 0.1031 - accuracy: 0.9748 - val_loss: 0.5422 - val_accuracy: 0.8947
Epoch 38/50
30/30 [==============================] - 2s 61ms/step - loss: 0.0980 - accuracy: 0.9797 - val_loss: 0.5264 - val_accuracy: 0.9001
Epoch 39/50
30/30 [==============================] - 2s 62ms/step - loss: 0.1261 - accuracy: 0.9763 - val_loss: 0.5430 - val_accuracy: 0.8990
Epoch 40/50
30/30 [==============================] - 2s 60ms/step - loss: 0.1074 - accuracy: 0.9812 - val_loss: 0.5371 - val_accuracy: 0.8985
Epoch 41/50
30/30 [==============================] - 2s 63ms/step - loss: 0.1227 - accuracy: 0.9750 - val_loss: 0.7442 - val_accuracy: 0.8835
Epoch 42/50
30/30 [==============================] - 2s 68ms/step - loss: 0.1619 - accuracy: 0.9658 - val_loss: 0.5752 - val_accuracy: 0.8942
Epoch 43/50
30/30 [==============================] - 2s 63ms/step - loss: 0.0976 - accuracy: 0.9779 - val_loss: 0.5898 - val_accuracy: 0.8969
Epoch 44/50
30/30 [==============================] - 2s 64ms/step - loss: 0.0979 - accuracy: 0.9807 - val_loss: 0.5929 - val_accuracy: 0.8980
Epoch 45/50
30/30 [==============================] - 2s 61ms/step - loss: 0.0952 - accuracy: 0.9820 - val_loss: 0.5846 - val_accuracy: 0.9044
Epoch 46/50
30/30 [==============================] - 2s 61ms/step - loss: 0.0933 - accuracy: 0.9803 - val_loss: 0.5844 - val_accuracy: 0.9028
Epoch 47/50
30/30 [==============================] - 2s 59ms/step - loss: 0.0742 - accuracy: 0.9826 - val_loss: 0.6086 - val_accuracy: 0.9012
Epoch 48/50
30/30 [==============================] - 2s 60ms/step - loss: 0.0910 - accuracy: 0.9825 - val_loss: 0.5470 - val_accuracy: 0.8937
Epoch 49/50
30/30 [==============================] - 2s 59ms/step - loss: 0.1027 - accuracy: 0.9774 - val_loss: 0.5620 - val_accuracy: 0.9049
Epoch 50/50
30/30 [==============================] - 2s 61ms/step - loss: 0.0755 - accuracy: 0.9848 - val_loss: 0.5978 - val_accuracy: 0.9066
233/233 [==============================] - 2s 7ms/step - loss: 0.0764 - accuracy: 0.9858
Training Accuracy: 0.9858
59/59 [==============================] - 0s 7ms/step - loss: 0.5978 - accuracy: 0.9066
Testing Accuracy:  0.9066
8/8 [==============================] - 0s 21ms/step - loss: 0.5978 - accuracy: 0.9066
Accuracy: 90.66%
Training Accuracy: 0.4108917925899547
Testing Accuracy: 0.3916973302822272
Mean Squared Error 0.10925886143931257
Testing Accuracy: 0.3916973302822272
```

```
Accuracy: 90.66%
Training Accuracy: 0.4108917925899547
Testing Accuracy: 0.3916973302822272
Mean Squared Error 0.10925886143931257
Testing Accuracy: 0.3916973302822272
```

a few example from results

| X | y (actual) | Predicted | |
|---|---|---|---|
| [ 950 1551 1922 81 1922 1581] | [0] | 1.0 | ✖ |
| [ 950 1551 1922 81 336 1352] | [1] | 1.0 | ✔ |
| [1451 1551 1922 1530 1922 1579] | [1] | 1.0 | ✔ |
| [ 950 1551 1922 81 336 1097] | [1] | 1.0 | ✔ |
| [ 950 1551 1922 81 1922 1485] | [0] | 1.0 | ✖ |
| [1451 1551 1922 1530 1922 1987] | [1] | 0.9 | ~ |
| [1075 1551 1922 663 1922 1514] | [1] | 1.0 | ✔ |
| [ 429 1551 336 1225 336 23] | [1] | 0.3 | ~ |
| [1451 1551 1922 1530 1922 423] | [1] | 1.0 | ✔ |

# Comparison Results for Successful Prediction

We can see that LSTM is more successful and sensitive in these calculations where we use the success status as the *y* variable. When we compared the results below, we can say that: When we search the answer of ‘predict if a project/campaign will be successful or not’, the LSTM algorithm works better than the CNN algorithm.

| Model | Accuracy |
|---|---|
| CNN | 76.91% |
| LSTM | 90.66% |

# Prediction of the Amount of Money Collected

We have done all the operations we mentioned in the previous slides for the "Pledged" value instead of "Successful State". Therefore, we don’t again explain the same steps. We want to add a few important details about pledged value. We use the pledged as the y variable. Our x variables are the same.

# Before the model..

```
kickstarter_X = []
kickstarter_y = []
for i, j in reduced_x_features.iterrows():
    tmp = str(reduced_x_features['launch_to_deadline_days'][i]) + " " + \
        str(reduced_x_features['staff_pick'][i]) + " " + \
        str(reduced_x_features['backers_count'][i]) + " " + \
        str(reduced_x_features['spotlight'][i]) + " " + \
        str(reduced_x_features['goal'][i]) + " " + \
        str(reduced_x_features['log_goal'][i]) + " " + \
        str(reduced_x_features['log_pledged'][i])
    kickstarter_X.append(tmp)
    kickstarter_y.append(reduced_y['pledged'][i])

max_words = 2000
max_length = 30
vector_length = 16

encoded_docs = [one_hot(d, max_words) for d in kickstarter_X]
padded_docs = pad_sequences(encoded_docs, maxlen=7, padding='post')

X_train, X_test, y_train, y_test = train_test_split(padded_docs,
        np.array(kickstarter_y)[:, None].astype(int), test_size=0.20,
        random_state=1234)
```

```
kickstarter_X
```

```
['60 0 5 0 100000.0 11.512925464970229 4.787491742782046',
 '32 0 17 0 3222.0 8.07775756373692 5.87493073085203',
 '30 0 12 0 13000.0 9.472704636443673 7.035268599281097',
 '30 0 7 0 6000.0 8.699514748210191 5.030437921392435',
 '30 0 5 0 7300.0 8.895629627136483 4.276666119016055',
 '60 0 10 0 10000.0 9.210340371976184 6.716594773520978',
 '45 0 7 0 10000.0 9.210340371976184 5.940171252720432',
 '60 0 3 0 2000.0 7.600902459542082 4.0943445622221',
 '20 0 11 0 2000.0 7.600902459542082 5.545177444479562',
 '20 0 18 0 5275.0 8.570733958344267 7.399398083331354',
 '30 0 4 0 1280.0 7.154615356913663 3.044522437723423',
 '30 0 4 0 20000.0 9.903487552536127 4.110873864173311',
 '28 0 35 0 4500.0 8.411832675758411 6.536691597591305',
 '30 0 6 0 450.0 6.109247582764365 3.433987204485146',
 '30 0 4 0 9000.0 9.104979856318357 3.784189633918261',
 '32 0 10 0 1200.0 7.090076835776092 6.1675164908883415',
 '40 0 14 0 1000.0 6.907755278982137 4.060443010546419',
 '30 0 3 0 1500.0 7.313220387090301 4.787491742782046',
 '30 0 3 0 2988.0 8.002359546252707 5.488937726156687',
 '30 0 6 0 1500.0 7.313220387090301 4.499809670330265',
 '14 0 4 0 2500.0 7.824046010856292 4.465908118654584',
 '50 0 7 0 10000.0 9.210340371976184 3.5553480614894135',
 '30 0 21 0 25000.0 10.126631103850338 7.531552381407289',
 '35 0 5 0 19778.0 9.892325487829936 3.7376696182833684',
 '31 0 4 0 15350.0 9.638870753015343 5.247024072160486',
```

```
kickstarter_y
```

```
[120.0,
 356.0,
 1136.0,
 153.0,
 72.0,
 826.0,
 380.0,
 60.0,
 256.0,
 1635.0,
 21.0,
 61.0,
 690.0,
 31.0,
 44.0,
 477.0,
 58.0,
 120.0,
 242.0,
 90.0,
 87.0,
 35.0,
 1866.0,
 42.0,
 190.0,
```

# Data Sets

```
X_train
array([[ 114, 1898,  114, ..., 1762,  348, 1423],
       [1090, 1740,  114, ...,  622,  348, 1448],
       [ 114,  835,  114, ..., 1709, 1425, 1724],
       ...,
       [ 114,  270,  114, ...,  327,  348,  339],
       [1090, 1179,  114, ..., 1060, 1782, 1401],
       [1090,  449,  114, ..., 1602, 1990,  547]], dtype=int32)
```

```
y_train
array([[3185],
       [4280],
       [  17],
       ...,
       [4276],
       [2025],
       [  50]])
```

```
X_test
array([[ 114, 1510,  114, ..., 1568,  491,  985],
       [ 114,  489,  114, ..., 1115, 1782,  126],
       [ 114, 1365,  114, ...,  289,  752,  493],
       ...,
       [1090, 1904,  114, ...,  855,  500,  685],
       [ 114, 1740,  114, ...,  622, 1782, 1284],
       [ 114,  971,  114, ...,  454,  500, 1742]], dtype=int32)
```

```
y_test
array([[ 143],
       [1956],
       [ 251],
       ...,
       [ 517],
       [1682],
       [1069]])
```

# Prediction of Amount of Money Collected with CNN

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 16)            32016
_________________________________________________________________
conv1d (Conv1D)              (None, 30, 32)            3616
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 15, 32)            0
_________________________________________________________________
dense (Dense)                (None, 15, 128)           4224
_________________________________________________________________
dropout (Dropout)            (None, 15, 128)           0
_________________________________________________________________
dense_1 (Dense)              (None, 15, 64)            8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 64)            0
_________________________________________________________________
dense_2 (Dense)              (None, 15, 32)            2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 32)            0
_________________________________________________________________
dense_3 (Dense)              (None, 15, 16)            528
_________________________________________________________________
dropout_3 (Dropout)          (None, 15, 16)            0
_________________________________________________________________
dense_4 (Dense)              (None, 15, 1)             17
=================================================================
Total params: 50,737
Trainable params: 50,737
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
30/30 [==============================] - 3s 41ms/step - loss: 7012.2432 - mse: 52122044.
Epoch 2/50
30/30 [==============================] - 0s 14ms/step - loss: -33951.3029 - mse: 4906339
Epoch 3/50
30/30 [==============================] - 0s 15ms/step - loss: -39036.9880 - mse: 6883361
Epoch 4/50
30/30 [==============================] - 0s 14ms/step - loss: -40133.3417 - mse: 7462120
Epoch 5/50
30/30 [==============================] - 0s 14ms/step - loss: -39407.0640 - mse: 6019801
Epoch 6/50
30/30 [==============================] - 0s 14ms/step - loss: -39875.8943 - mse: 5911995
Epoch 7/50
```

```
30/30 [==============================] - 0s 14ms/step - loss: -38911.0333 - mse: 63873894.1935 - val_loss: -42639.6445 - val_mse: 84397568.0000
Epoch 34/50
30/30 [==============================] - 0s 14ms/step - loss: -39328.6124 - mse: 50378846.0645 - val_loss: -42639.6445 - val_mse: 84397368.0000
Epoch 35/50
30/30 [==============================] - 0s 14ms/step - loss: -38686.5318 - mse: 67549320.7742 - val_loss: -42639.6445 - val_mse: 84396912.0000
Epoch 36/50
30/30 [==============================] - 0s 14ms/step - loss: -39443.7332 - mse: 66609502.1935 - val_loss: -42639.6445 - val_mse: 84396880.0000
Epoch 37/50
30/30 [==============================] - 0s 14ms/step - loss: -37166.9146 - mse: 38197272.6452 - val_loss: -42639.6445 - val_mse: 84396776.0000
Epoch 38/50
30/30 [==============================] - 0s 16ms/step - loss: -38769.1003 - mse: 63156037.1613 - val_loss: -42639.6445 - val_mse: 84396360.0000
Epoch 39/50
30/30 [==============================] - 0s 14ms/step - loss: -40014.5530 - mse: 69390910.0323 - val_loss: -42639.6445 - val_mse: 84396256.0000
Epoch 40/50
30/30 [==============================] - 0s 14ms/step - loss: -40235.8672 - mse: 68055363.7419 - val_loss: -42639.6445 - val_mse: 84396256.0000
Epoch 41/50
30/30 [==============================] - 0s 14ms/step - loss: -41700.9685 - mse: 90571465.1613 - val_loss: -42639.6445 - val_mse: 84396216.0000
Epoch 42/50
30/30 [==============================] - 0s 14ms/step - loss: -38682.0558 - mse: 53485839.2258 - val_loss: -42639.6445 - val_mse: 84396168.0000
Epoch 43/50
30/30 [==============================] - 0s 14ms/step - loss: -39001.4313 - mse: 66508281.4194 - val_loss: -42639.6445 - val_mse: 84396136.0000
Epoch 44/50
30/30 [==============================] - 0s 14ms/step - loss: -39277.8934 - mse: 49464350.8387 - val_loss: -42639.6445 - val_mse: 84395864.0000
Epoch 45/50
30/30 [==============================] - 0s 14ms/step - loss: -38902.3553 - mse: 52800598.9677 - val_loss: -42639.6445 - val_mse: 84395760.0000
Epoch 46/50
30/30 [==============================] - 0s 14ms/step - loss: -40241.2186 - mse: 68309021.6774 - val_loss: -42639.6445 - val_mse: 84395728.0000
Epoch 47/50
30/30 [==============================] - 0s 15ms/step - loss: -38966.3041 - mse: 53757656.7097 - val_loss: -42639.6445 - val_mse: 84395248.0000
Epoch 48/50
30/30 [==============================] - 0s 14ms/step - loss: -39467.3148 - mse: 53549221.2903 - val_loss: -42639.6445 - val_mse: 84393600.0000
Epoch 49/50
30/30 [==============================] - 0s 14ms/step - loss: -38736.0798 - mse: 56612921.4194 - val_loss: -42639.6445 - val_mse: 84392944.0000
Epoch 50/50
30/30 [==============================] - 0s 14ms/step - loss: -39959.7795 - mse: 58891769.5484 - val_loss: -42639.6445 - val_mse: 84392744.0000
8/8 [==============================] - 0s 4ms/step - loss: -42639.6445 - mse: 84392744.0000
Training Accuracy: 76.79439365767561
Testing Accuracy: 65.66132636854505
Mean Squared Error 26312415.842544302
```

```
Training Accuracy: 76.79439365767561
Testing Accuracy: 65.66132636854505
Mean Squared Error 26312415.842544302
```

a few
example
from results

| X | y (actual) | Predicted |
|---|---|---|
| [ 114 135 114 348 971 1990 1103] | [47] | 34.0 |
| [1090 504 114 752 55 752 189] | [283] | 258.15 |
| [114 621 114 348 864 491 430] | [140] | 124.25 |
| [ 114 1836 114 1371 1933 1951 334] | [16652] | 11144.8 |
| [1090 1612 114 449 1438 449 1098] | [40502] | 34024.85 |
| [ 114 234 114 348 92 500 1045] | [875] | 681.35 |
| [ 114 1212 114 207 933 1990 1125] | [37] | 34.0 |
| [ 114 822 114 207 360 491 1570] | [63] | 100.7 |
| [1090 1651 114 752 1690 752 1690] | [200] | 251.8 |

# Prediction of Amount of Money Collected with LSTM

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 30, 16)            32016
_________________________________________________________________
lstm (LSTM)                  (None, 30, 128)           74240
_________________________________________________________________
dropout_4 (Dropout)          (None, 30, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 64)            49408
_________________________________________________________________
dropout_5 (Dropout)          (None, 30, 64)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 30, 32)            12416
_________________________________________________________________
dropout_6 (Dropout)          (None, 30, 32)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 16)                3136
_________________________________________________________________
dropout_7 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
=================================================================
Total params: 171,233
Trainable params: 171,233
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
30/30 [==============================] - 11s 128ms/step - loss: 11923.8557
Epoch 2/50
30/30 [==============================] - 2s 76ms/step - loss: 5289.5608 -
Epoch 3/50
30/30 [==============================] - 2s 74ms/step - loss: 1593.6903 -
Epoch 4/50
30/30 [==============================] - 2s 74ms/step - loss: -38937.6896
Epoch 5/50
30/30 [==============================] - 2s 74ms/step - loss: -40111.3062
Epoch 6/50
```

```
Epoch 38/50
30/30 [==============================] - 2s 79ms/step - loss: -39037.1896 - mse: 55931344.7742 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 39/50
30/30 [==============================] - 3s 84ms/step - loss: -38702.5566 - mse: 50347617.4194 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 40/50
30/30 [==============================] - 2s 79ms/step - loss: -37651.7639 - mse: 43596262.7742 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 41/50
30/30 [==============================] - 2s 81ms/step - loss: -38573.0602 - mse: 54955791.8710 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 42/50
30/30 [==============================] - 2s 78ms/step - loss: -39125.1707 - mse: 51789280.0000 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 43/50
30/30 [==============================] - 2s 74ms/step - loss: -40988.2738 - mse: 61156648.6452 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 44/50
30/30 [==============================] - 2s 75ms/step - loss: -39300.6195 - mse: 64449872.3226 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 45/50
30/30 [==============================] - 2s 77ms/step - loss: -38238.9680 - mse: 47145740.5806 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 46/50
30/30 [==============================] - 2s 75ms/step - loss: -39193.2673 - mse: 51795553.0323 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 47/50
30/30 [==============================] - 2s 73ms/step - loss: -39105.0823 - mse: 62160863.7419 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 48/50
30/30 [==============================] - 2s 74ms/step - loss: -39646.0974 - mse: 52674602.5806 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 49/50
30/30 [==============================] - 2s 73ms/step - loss: -38502.8212 - mse: 55606054.3226 - val_loss: -42639.6445 - val_mse: 84423568.0000
Epoch 50/50
30/30 [==============================] - 2s 76ms/step - loss: -37963.1094 - mse: 47661201.1613 - val_loss: -42639.6445 - val_mse: 84423568.0000
233/233 [==============================] - 2s 9ms/step - loss: -39042.2773 - mse: 55941444.0000
Training Accuracy: 55941444.0000
59/59 [==============================] - 1s 9ms/step - loss: -42639.6523 - mse: 84423560.0000
Testing Accuracy:  84423560.0000
8/8 [==============================] - 0s 27ms/step - loss: -42639.6445 - mse: 84423568.0000
Training Accuracy: 96.48762596194376
Testing Accuracy: 92.12900602520273
Mean Squared Error 6031242.463870651
```

```
Training Accuracy: 96.48762596194376
Testing Accuracy: 92.1290060252027З
Mean Squared Error 6031242.463870651
```

a few
example
from results

| X | y (actual) | Predicted |
|---|---|---|
| [ 114 135 114 348 971 1990 1103] | [47] | 31.63752928644034 |
| [1090 504 114 752 55 752 189] | [283] | 241.6743412970838 |
| [114 621 114 348 864 491 430] | [140] | 104.0141603199677 |
| [ 114 1836 114 1371 1933 1951 334] | [16652] | 12476.59626463691 |
| [1090 1612 114 449 1438 449 1098] | [40502] | 33232.3365437678 |
| [ 114 234 114 348 92 500 1045] | [875] | 702.193723068181 |
| [ 114 1212 114 207 933 1990 1125] | [37] | 31.63752928644034 |
| [ 114 822 114 207 360 491 1570] | [63] | 99.4361856507565 |
| [1090 1651 114 752 1690 752 1690] | [200] | 241.6743412970838 |

# Comparison Results for Amount of Money Collected Prediction

We can see that LSTM is more successful and sensitive in these calculations where we use the pledged as the *y* variable. When we compared the results below, we can say that: When we search the answer of 'predict the amount of money collected', the LSTM algorithm works better than the CNN algorithm. In addition, we observed that LSTM gives more sensitive results.

| Model | Accuracy |
| --- | --- |
| CNN | 65.66% |
| LSTM | 92.13% |

# Utilized Resources

Keras: Multiple Inputs and Mixed Data

House Price Prediction using Machine Learning

Long Short Term Memory (LSTM)

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

CNN Long Short-Term Memory Networks

Convolutional Neural Networks in Python with Keras

How to Make Predictions with Keras

How to Make Predictions with scikit-learn

Text classification using CNN

Dense neural netLSTM and CNN on IMDB