# Prosperity Prognosticator:Machine Learning for Startup Success Prediction Display Portal

A ServiceNow Admin Project

### Team Details

**Team ID: LTVIP2026TMIDS87262**
**Team Size:** 4
**Team Leader**: Kummaragunta Jahnavi Priya
**Team Members:**
M Janani Janani
Naresh V
Poola Tharun

## Abstract

The escalating global challenge of waste management necessitates innovative, efficient, and sustainable solutions. Current waste sorting and recycling processes often rely on manual labor or conventional machine learning methods, which can be slow, error-prone, and inefficient at handling the increasing complexity and volume of mixed municipal solid waste (MSW). This research addresses these limitations by introducing a novel framework that leverages Transfer Learning (TL) within the field of Clean Technology to significantly enhance the accuracy and speed of automated waste classification and sorting, thereby promoting a circular economy. This clean tech application not only optimizes recycling operations but also contributes directly to mitigating landfill dependency and reducing greenhouse gas emissions. In conclusion, this research validates Transfer Learning as a powerful, practical, and immediately deployable technology for achieving intelligent, sustainable, and scalable waste management solutions globally. The developed TL-based classification system is directly applicable for deployment in automated Material Recovery Facilities (MRFs), enabling a more efficient, cost-effective, and environmentally responsible sorting process.

# Introduction

In response, the field of **Clean Technology (Clean Tech)** has emerged as a crucial driver for sustainable solutions. Clean Tech encompasses technologies designed to enhance environmental sustainability through practices like energy efficiency, renewable energy, and, critically, **advanced resource recovery**. For waste management, this transition mandates a shift from a linear "take-make-dispose" model to a sophisticated **Circular Economy** framework, where maximizing the recovery of clean, high-purity materials is paramount.

# Project Overview

This project aims to simplify **Prosperity Prognosticator**, Key goals:

- **Dramatically increase sorting purity**, leading to higher-quality recycled materials.
- **Enhance operational efficiency** and reduce the high costs associated with manual and error-prone sorting.
- **Accelerate the transition to a Circular Economy**, by turning waste from a liability into a valuable manufacturing feedstock.
- By integrating Transfer Learning into automated sorting infrastructure, this work demonstrates a powerful pathway for Clean Tech to deliver truly sustainable, scalable, and economically viable solutions to the global waste criteria.

## Problem Statement

In many organizations and industries, large volumes of items (such as files, products, waste, or data) need to be sorted efficiently. Traditional manual or basic automated sorting methods are often slow, error-prone, and unable to adapt to changing requirements.

The **Smart Sorting System** aims to develop an intelligent solution that automatically identifies, classifies, and sorts items based on predefined features such as size, type, colour, priority, or content using modern technologies like sensors, artificial intelligence, and automation.

The system should improve accuracy, reduce human effort, save time, and increase overall productivity while ensuring flexibility and scalability.

# Brainstorming & Module Planning

Modules identified:

- **Creating a function for evaluation**
- **Training and testing the Models using multiple algorithms**
- **Performance Testing & Hyperparameter Tuning**
- **Testing model with multiple evaluation metrics**
- **Comparing model accuracy before & after applying hyperparameter tuning**
- **Comparing model accuracy for different numbers of features.**
- **Building a model with appropriate features.**

**Model Deployment:**

- **Save the best model**
- **Integrate with Web Framework**

# • Requirement Analysis

## Functional Requirements:

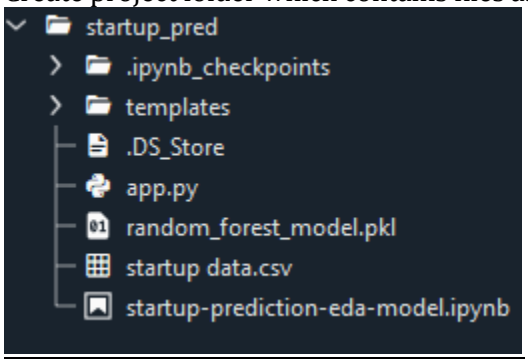
- Create a menu with fields (Date, Meal, Food Items, Special Note)
- Submit via catalog item
- Approve/reject by cafeteria manager
- Publish/unpublish menu using UI Action
- View menus via the Service Portal

## ⚙ Non-Functional Requirements:

- Must be easy-to-use via ServiceNow UI
- Real-time publishing and notification
- Access control by roles (Admin, Manager, Employee)

# Project Implementation

Create project folder which contains files as shown below:




- The data obtained is in csv files, for training and testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- random_forest model. pkl is the saved model. This will further be used in the Flask integration.

Collect the dataset
There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.
Link: https://www.kaggle.com/datasets/manishkc06/startup-success-prediction
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.
Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

## Importing the Libraries

```
In [1]: import numpy as np # linear algebra
        import pandas as pd # data processing
        pd.set_option('display.max_columns', None)
        import os
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import GridSearchCV
        from sklearn.metrics import classification_report
```

Read the Dataset
Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.
In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
data = pd.read_csv('startup data.csv')
```

```
data.head()
```

| | Unnamed: 0 | state_code | latitude | longitude | zip_code | id | city | Unnamed: 6 | name | labels | founded_at | closed_at | first_funding_at | last_fun |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1005 | CA | 42.358880 | -71.056820 | 92101 | c:6669 | San Diego | NaN | Bandsintown | 1 | 1/1/2007 | NaN | 4/1/2009 | |
| 1 | 204 | CA | 37.238916 | -121.973718 | 95032 | c:16283 | Los Gatos | NaN | TriCipher | 1 | 1/1/2000 | NaN | 2/14/2005 | 12/ |
| 2 | 1001 | CA | 32.901049 | -117.192656 | 92121 | c:65620 | San Diego | San Diego CA 92121 | Plixi | 1 | 3/18/2009 | NaN | 3/30/2010 | 3/ |
| 3 | 738 | CA | 37.320309 | -122.050040 | 95014 | c:42668 | Cupertino | Cupertino CA 95014 | Solidcore Systems | 1 | 1/1/2002 | NaN | 2/17/2005 | 4/ |
| 4 | 1002 | CA | 37.779281 | -122.419236 | 94105 | c:65806 | San Francisco | San Francisco CA 94105 | Inhale Digital | 0 | 8/1/2010 | 10/1/2012 | 8/1/2010 | |

## Visualisation

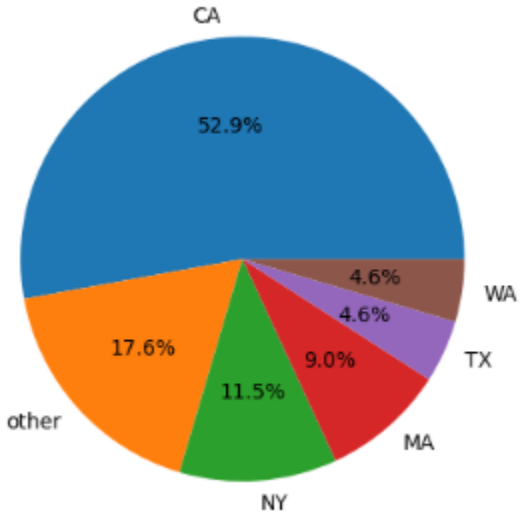
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

## Univariate & Multivariate Analysis

```
data['State'] = 'other'
data.loc[(data['state_code'] == 'CA'), 'State'] = 'CA'
data.loc[(data['state_code'] == 'NY'), 'State'] = 'NY'
data.loc[(data['state_code'] == 'MA'), 'State'] = 'MA'
data.loc[(data['state_code'] == 'TX'), 'State'] = 'TX'
data.loc[(data['state_code'] == 'WA'), 'State'] = 'WA'
```

```
state_count = data['State'].value_counts()
plt.pie(state_count, labels = state_count.index, autopct = '%1.1f%%')
plt.show()
```

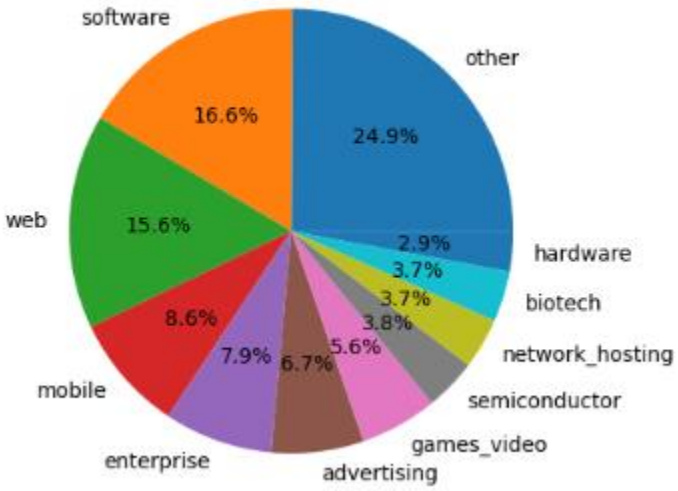
It assigns the value 'other' to the 'State' column for all rows in the 'data' DataFrame. Then, specific rows where the 'state_code' matches certain values (CA, NY, MA, TX, WA) are assigned their respective state names in the 'State' column. The 'value_counts()' method is used to count the occurrences of each unique value in the 'State' column, and a pie chart is created to visualize the distribution of states.




```
data['category'] = 'other'
data.loc[(data['category_code'] == 'software'), 'category'] = 'software'
data.loc[(data['category_code'] == 'web'), 'category'] = 'web'
data.loc[(data['category_code'] == 'mobile'), 'category'] = 'mobile'
data.loc[(data['category_code'] == 'enterprise'), 'category'] = 'enterprise'
data.loc[(data['category_code'] == 'advertising'), 'category'] = 'advertising'
data.loc[(data['category_code'] == 'games_video'), 'category'] = 'games_video'
data.loc[(data['category_code'] == 'semiconductor'), 'category'] = 'semiconductor'
data.loc[(data['category_code'] == 'network_hosting'), 'category'] = 'network_hosting'
data.loc[(data['category_code'] == 'biotech'), 'category'] = 'biotech'
data.loc[(data['category_code'] == 'hardware'), 'category'] = 'hardware'
```

```
category_count = data['category'].value_counts()
plt.pie(category_count, labels = category_count.index, autopct = '%1.1f%%')
plt.show()
```

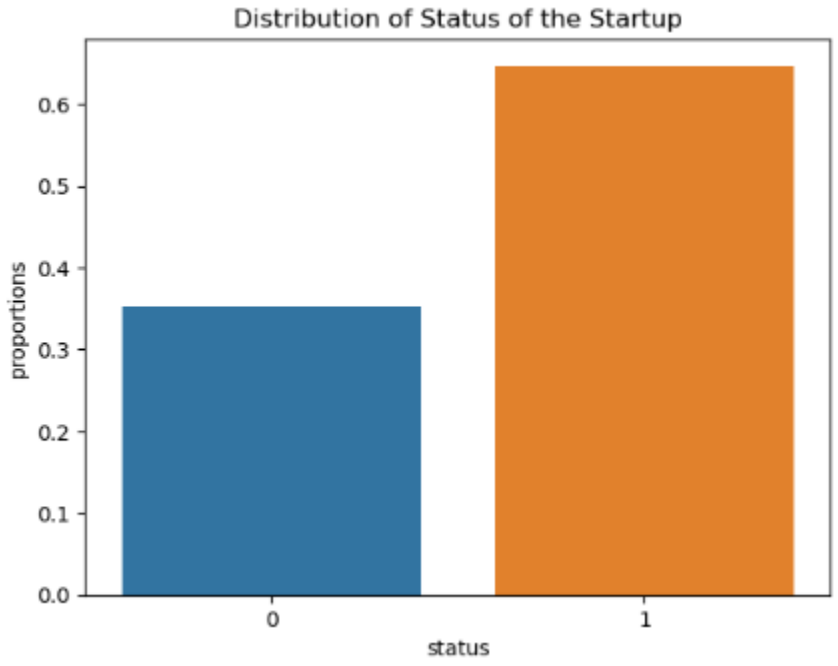
The given code snippet performs a univariate analysis. It assigns the value 'other' to the 'category' column for all rows in the 'data' DataFrame. Then, specific rows matching certain 'category_code' values are assigned corresponding categories in the 'category' column. The 'value_counts ()' method is used to count the occurrences of each unique category in the 'category' column, and a pie chart is created to visualize the distribution of categories.




- **Distribution of Status of Startup**

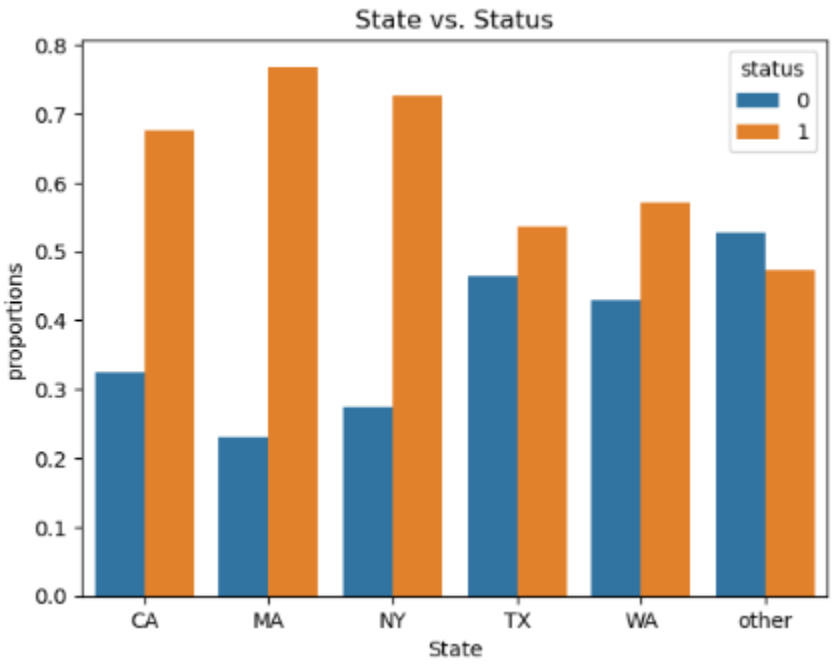
```
prop_df = data.groupby('status').size().reset_index(name = 'counts')
prop_df['proportions'] = prop_df['counts']/prop_df['counts'].sum()
```

```
sns.barplot(data = prop_df, x = 'status', y = 'proportions')
plt.title('Distribution of Status of the Startup')
```



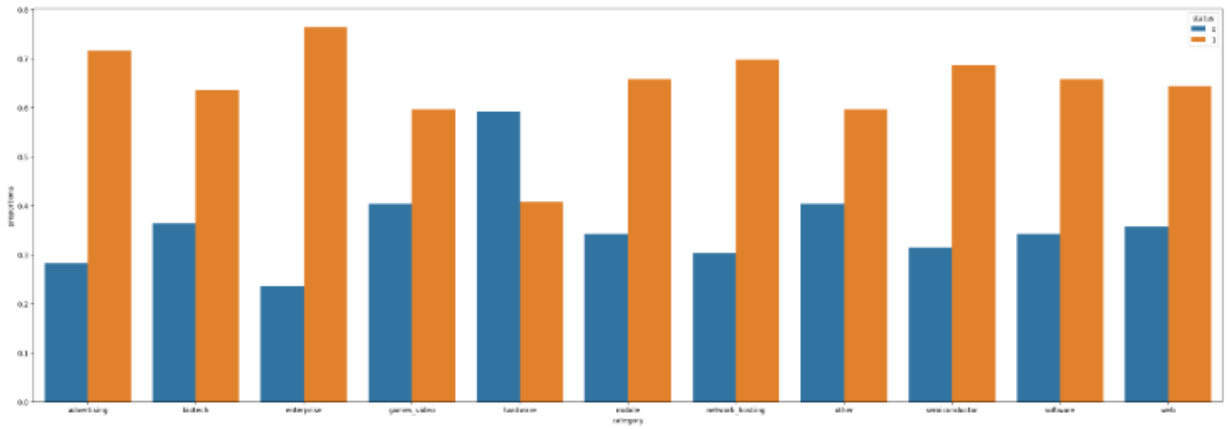

- **State Vs. Staus**

```
prop_df = data.groupby(['State','status'],group_keys = True).size().reset_index(name='count')
prop_df['proportions'] = prop_df.groupby('State')['count'].apply(lambda x: x/x.sum())
sns.barplot(data = prop_df, x = 'State', y = 'proportions',hue = 'status')
plt.title('State vs. Status')
```




- **State Vs. Category**

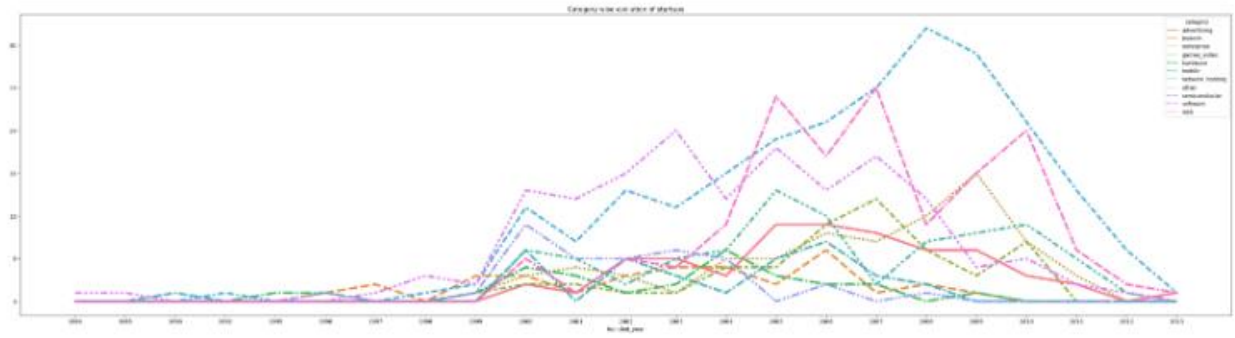
```
fig, ax = plt.subplots(figsize = (30,10))
prop_df = data.groupby(['category','status']).size().reset_index(name='counts')
prop_df['proportions'] = prop_df.groupby('category')['counts'].apply(lambda x: x/float(x.sum()))
sns.barplot(data = prop_df, x = 'category', y = 'proportions',hue = 'status')
```



- **Category Vs. Founded- Year**

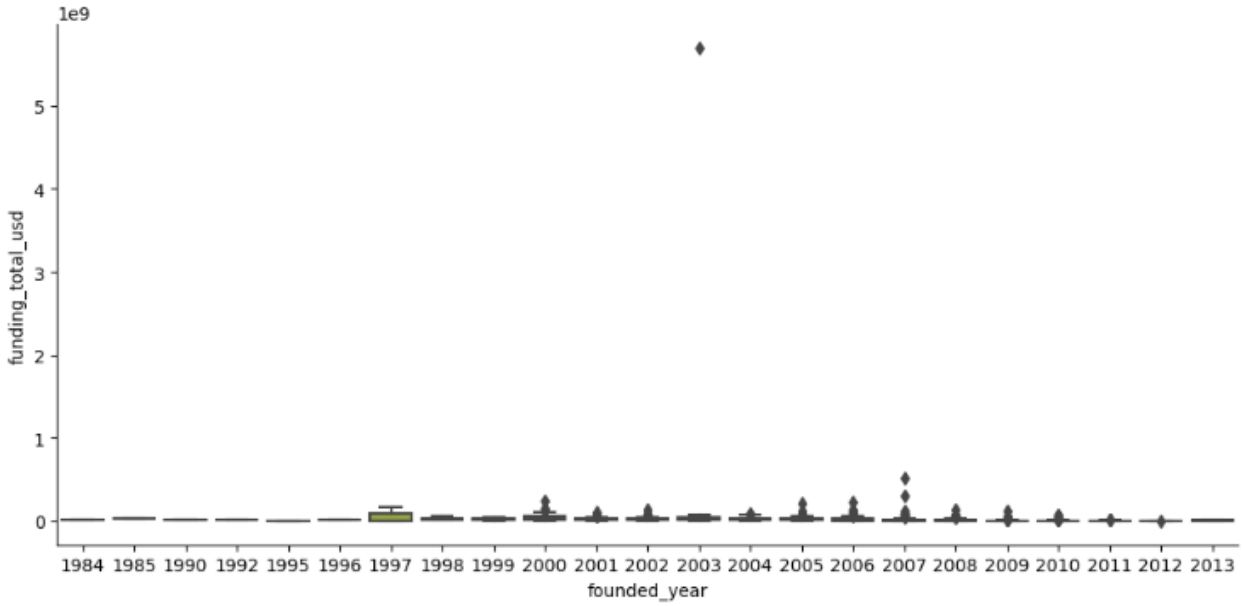
```
cat_year = pd.crosstab(index = data['founded_year'], columns = data['category'])
```

```
fig, ax = plt.subplots(figsize=(40,10))
sns.lineplot(data = cat_year, lw = 4)
plt.title('Category wise evolution of startups')
```



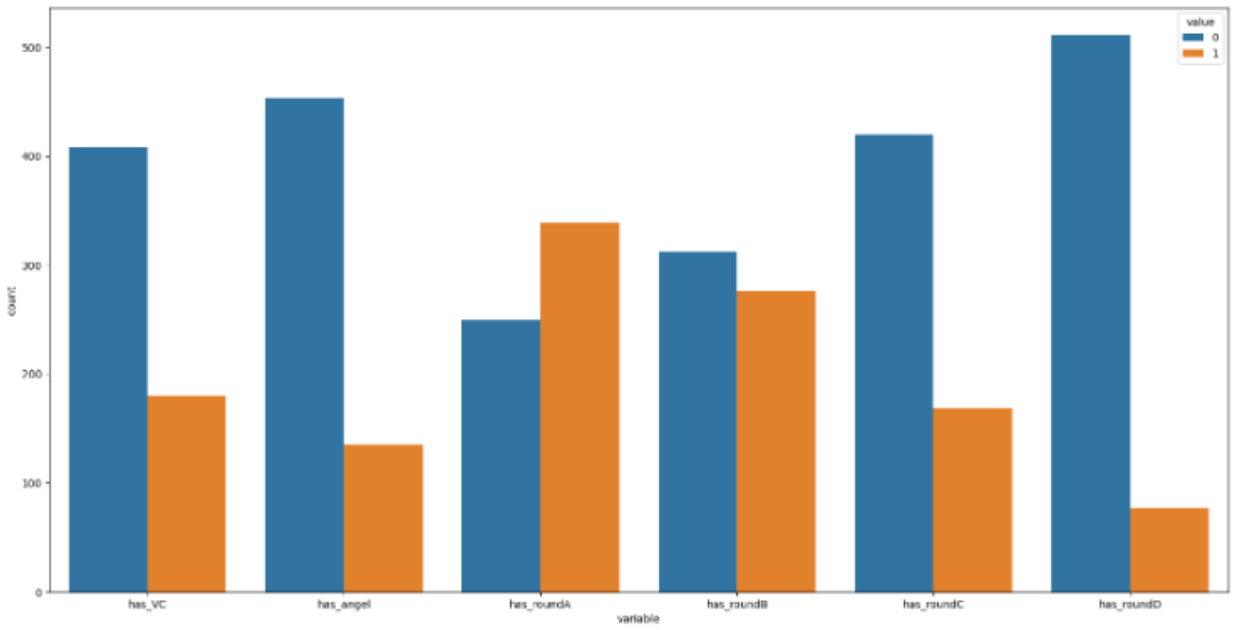
- **Founded- Year Vs. Total Funding**

```
sns.catplot(data=data, x="founded_year", y="funding_total_usd",
            kind="box",height=5, aspect=2,order = ['1984','1985','1990','1992','1995',
                                                   '1996','1997','1998','1999','2000',
                                                   '2001','2002','2003','2004','2005',
                                                   '2006','2007','2008','2009','2010',
                                                   '2011','2012','2013'])
```




- **Has_VC, Has_angel, Has_roundA, Has_roundB, Has_roundC, Has_roundD**

```
fig, ax = plt.subplots(figsize = (20,10))
d = data.loc[data['status'] == '1']
f = d[["has_VC","has_angel","has_roundA","has_roundB","has_roundC","has_roundD"]]
sns.countplot(data = pd.melt(f), x = 'variable', hue = 'value')
```




- **Statistical Analysis**

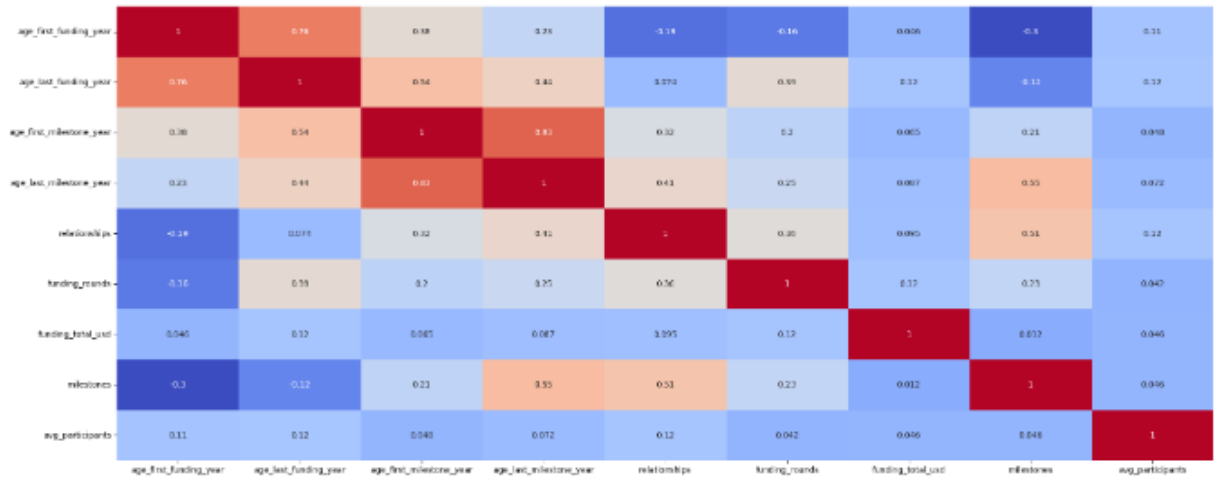
```
data.describe(include = ['float64','int64'])
```

| | age_first_funding_year | age_last_funding_year | age_first_milestone_year | age_last_milestone_year | relationships | funding_rounds | funding_total_usd | milest |
|---|---|---|---|---|---|---|---|---|
| count | 914.000000 | 914.000000 | 914.000000 | 914.000000 | 914.000000 | 914.000000 | 9.140000e+02 | 914.00 |
| mean | 2.270677 | 3.947489 | 2.613675 | 3.985237 | 7.680525 | 2.312910 | 2.552193e+07 | 1.83 |
| std | 2.457724 | 2.930272 | 2.816996 | 3.382481 | 7.265480 | 1.394131 | 1.905590e+08 | 1.32 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.100000e+04 | 0.00 |
| 25% | 0.568475 | 1.669175 | 0.000000 | 1.000000 | 3.000000 | 1.000000 | 2.712500e+06 | 1.00 |
| 50% | 1.449350 | 3.545200 | 2.000000 | 3.754800 | 5.000000 | 2.000000 | 1.000000e+07 | 2.00 |
| 75% | 3.578075 | 5.558900 | 4.002700 | 6.026050 | 10.000000 | 3.000000 | 2.485264e+07 | 3.00 |
| max | 21.895900 | 21.895900 | 24.684900 | 24.684900 | 63.000000 | 10.000000 | 5.700000e+09 | 8.00 |

- **Correlation Plot**

```
fig, ax = plt.subplots(figsize = (30,10))
corr = data.select_dtypes(include=['int64','float64']).corr()
sns.heatmap(corr, cmap = 'coolwarm', annot = True)
```

```
<Axes: >
```



## Pre- Processing

- ## Reducing the Number of Categories

```
print(data['state_code'].equals(data['state_code.1']))
```

```
False
```

```
df = data.loc[data['state_code'] != data['state_code.1']]
df.style.set_properties(**{'background-color': 'yellow'}, subset=['state_code','state_code.1'])
```

| | state_code | city | labels | founded_at | closed_at | first_funding_at | last_funding_at | age_first_funding_year | age_last_funding_year |
|---|---|---|---|---|---|---|---|---|---|
| 515 | CA | Menlo Park | 0 | 1/1/2005 | 9/1/2010 | 3/1/2007 | 4/15/2008 | 2.161600 | 3.287700 |

```
state = data['state_code'].value_counts().to_frame()
state['proportion'] = state['state_code']/sum(state['state_code'])*100
state
```

The code provided compares the 'state_code' column with the 'state_code.1' column in the 'data' DataFrame. It prints whether the two columns are equal. It then creates a new DataFrame 'df' by selecting rows where the 'state_code' column is not equal to the 'state_code.1' column. The 'df' DataFrame is styled with a yellow background for the 'state_code' and 'state_code.1' columns. Additionally, the code calculates the count of each unique value in the 'state_code' column and creates a new DataFrame 'state' to store the counts. The 'proportion' column is added to 'state', which represents the proportion of each state code count out of the total count, expressed as a percentage.

| | state_code | proportion |
|---|---|---|
| CA | 488 | 52.871073 |
| NY | 106 | 11.484290 |
| MA | 83 | 8.992416 |
| TX | 42 | 4.550379 |
| WA | 42 | 4.550379 |
| CO | 19 | 2.058505 |
| IL | 18 | 1.950163 |
| PA | 17 | 1.841820 |
| VA | 13 | 1.408451 |
| GA | 11 | 1.191766 |
| NC | 7 | 0.758397 |
| OR | 7 | 0.758397 |
| NJ | 7 | 0.758397 |
| MD | 7 | 0.758397 |
| FL | 6 | 0.650054 |
| OH | 6 | 0.650054 |
| MN | 5 | 0.541712 |
| DC | 4 | 0.433369 |
| CT | 4 | 0.433369 |
| TN | 3 | 0.325027 |

State Code Variable- Check whether the state_code and state_code.1 variables contain the same data for each row - One row is not equal to each other, because in state_code.1 variable is a missing value. Therefore we assume the correct column is state_code (considering the city column).
Reduce the number of categories - The top 5 categories of the state_code variable cover more than 80% of all the data. Therefore the other categories apart from the top 5 (CA, NY, MA, TX, WA) are considered as 'other'.

- ## Dropping the Irrelevant Columns

```
data = data.drop(['category_code','is_software','is_web','is_mobile','is_enterprise','is_advertising',
                  'is_gamesvideo','is_ecommerce','is_biotech','is_consulting','is_othercategory'],axis = 1)
```

The code snippet drops multiple columns ('category_code', 'is_software', 'is_web', 'is_mobile', 'is_enterprise', 'is_advertising', 'is_gamesvideo', 'is_ecommerce', 'is_biotech', 'is_consulting', 'is_othercategory') from the 'data' DataFrame, effectively removing those columns from the dataset.

## Train-Test Split

The code splits the DataFrame df into input features X and the target variable y. The target variable is assigned to y, while the input features are assigned to X after removing the 'Adaptivity Level' column using the drop() function. The data is then split into training and testing sets using the train_test_split() function, with 70% of the data allocated for training (X_train, y_train) and 30% for testing (X_test, y_test), ensuring reproducibility with a random state of 116.

```
#split train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.3, random_state = 0)
```

## Define Pre-Processing and Modelling Function

- The code snippet performs a grid search using the GridSearchCV class to find the best combination of hyperparameters for a Random Forest classifier. The parameter grid is defined with different values for 'n_estimators', 'max_depth', 'min_samples_split', 'min_samples_leaf', and 'bootstrap'. The grid search is performed using 5-fold cross-validation (cv=5) and parallelized (-1 for n_jobs). After fitting the grid search object to the training data, it prints the best parameters found based on the evaluation of different parameter combinations.

```
#rf = RandomForestClassifier()
param_grid = {'n_estimators':[100,200,300],
              'max_depth':[10,20,30],
              'min_samples_split':[2,4,6],
              'min_samples_leaf':[1,2,3],
              'bootstrap':[True,False]}

grid_search = GridSearchCV(estimator=rf, param_grid = param_grid, cv = 5, n_jobs = -1, verbose = False)
grid_search.fit(x_train, y_train)

print('Best parameters:', grid_search.best_params_)
```

```
Best parameters: {'bootstrap': False, 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

- The _scale function takes in training and validation data along with a list of features. It applies the StandardScaler to standardize the numerical features in the training data. It then uses the computed scaler to transform both the training and validation data. The function returns the updated training and validation data with the scaled features. This ensures that the features have zero mean and unit variance, which can be beneficial for certain machine learning algorithms.

```
#rf = RandomForestClassifier(n_estimators=100,bootstrap=False,max_depth=20,min_samples_leaf=2, min_samples_split=2)
model_rf = rf.fit(x_train,y_train)
y_pred_rf = model_rf.predict(x_test)
cr_rf = classification_report(y_pred_rf, y_test)
print(cr_rf)
```

```
              precision    recall  f1-score   support

           0       0.59      0.78      0.67        68
           1       0.92      0.82      0.87       207

    accuracy                           0.81       275
   macro avg       0.75      0.80      0.77       275
weighted avg       0.84      0.81      0.82       275
```

The classification report provides an evaluation of the model's performance. For class 0, the precision is 0.59, recall is 0.78, and F1-score is 0.67. For class 1, the precision is 0.92, recall is 0.82, and F1-score is 0.87. The overall accuracy of the model is 0.81. The macro average of precision, recall, and F1-score is 0.75, 0.80, and 0.77, respectively. The weighted average of precision, recall, and F1-score is 0.84, 0.81, and 0.82, respectively.

## Analyzing Accuracy

The code snippet performs evaluation and saves the results of a model. It imports several metrics from sklearn.metrics, including accuracy_score, classification_report, confusion_matrix, roc_curve, and roc_auc_score. The model's predictions are compared with the true labels (y_test), and the accuracy score is calculated. Additionally, a classification report is generated using the predicted labels and the true labels. Finally, the results, including the predictions, accuracy, and classification report, are stored in a dictionary called "results**."**

```
#applying Random forest classifier
mode=RandomForestClassifier()
model.fit(x_train, y_train)
y_pred_test=model.predict(x_test)
y_pred_train=model.predict(x_train)
```

```
#checking accuracy
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
test_acc = accuracy_score(y_test,y_pred_test)
train_acc = accuracy_score(y_train,y_pred_train)
print('test_acc: ', test_acc)
print('train_acc: ', train_acc)
```

```
test_acc:  0.8
train_acc:  1.0
```

The Random Forest classifier achieved an accuracy of 80% on the test dataset (test_acc). This means that the model correctly predicted the target variable for 80% of the test instances. However, the model achieved a perfect accuracy of 100% on the training dataset (train_acc), indicating that it may be overfitting the training data. Overfitting occurs when a model performs exceptionally well on the training data but fails to generalize well to unseen data. Further evaluation and tuning may be necessary to address the potential overfitting issue.

## Create a new app.py file which will be stored in the Flask folder.

- Import the necessary Libraries.

```
from flask import Flask, render_template, request
import joblib
```

- This code uses the joblib module to load a saved machine-learning model and a saved preprocessing object. The joblib.load function is used to load the model object from a file called model.pkl and the scaler object from a file called random_forest_model.pkl. These objects are then stored in the model and scaler variables, respectively. The loaded model object can be used to make predictions on new data and the loaded scaler object can be used to preprocess the data in the same way as it was done during training. This process of loading saved models and preprocessing objects can save time and resources when working on a new project or dataset.

```
app = Flask(__name__)
model = joblib.load('random_forest_model.pkl')
```

- This code creates a new instance of a Flask web application using the Flask class from the Flask library. The __name__ argument specifies the name of the application's module **or package.**

```
app = Flask(__name__)
```

- The code snippet defines a route '/' for the Flask application. When a user accesses the homepage of the application, it calls the 'home' function. This function renders the 'index.html' template, which will be displayed as the homepage of the application.

```
@app.route('/')
def home():
    return render_template('index.html')
```

- The code snippet defines a route '/predict' for the Flask application, which expects a POST request. When this route is accessed, the 'predict' function is called. Inside the function, it retrieves input values from a form submitted with the POST request. The values are converted to float and stored in respective variables for further processing.

```
@app.route('/predict', methods=['POST'])
def predict():
    # Get the input values from the form
    age_first_funding_year = float(request.form['age_first_funding_year'])
    age_last_funding_year = float(request.form['age_last_funding_year'])
    age_first_milestone_year = float(request.form['age_first_milestone_year'])
    age_last_milestone_year = float(request.form['age_last_milestone_year'])
    relationships = float(request.form['relationships'])
    funding_rounds = float(request.form['funding_rounds'])
    funding_total_usd = float(request.form['funding_total_usd'])
    milestones = float(request.form['milestones'])
    avg_participants = float(request.form['avg_participants'])
```

- The code snippet creates a list called 'input_data' containing the input values retrieved from the form. It then uses a loaded model to make a prediction based on the input data. The predicted label is mapped to a meaningful output ('Acquired' or 'Closed'). Finally, the 'result' variable is passed to the 'result.html' template for rendering the prediction result on a webpage.

```
# Create a list with the input values
input_data = [
    age_first_funding_year,
    age_last_funding_year,
    age_first_milestone_year,
    age_last_milestone_year,
    relationships,
    funding_rounds,
    funding_total_usd,
    milestones,
    avg_participants
]

# Make a prediction using the loaded model
prediction = model.predict([input_data])[0]

# Map the predicted label to a meaningful output
if prediction == 1:
    result = 'Acquired'
else:
    result = 'Closed'

# Render the prediction result
return render_template('result.html', result=result)
```

### Main Function:

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if __name__ == '__main__': line checks if the script is the main module being executed and if so, runs the Flask application using the app.run() method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

```
if __name__ == '__main__':
    app.run()
```

# Advantages & Disadvantages

### Advantages:

- Better Decision Making
- Early Risk Detection
- Resource Optimization
- Approval and record tracking

### ⚠ Disadvantages:

- Dependence on Data Quality
- High Development Cost
- Privacy Concerns

# Conclusion

The Prosperity Prognosticator is an intelligent system designed to predict future growth, success, or financial outcomes using data analysis and advanced technologies. It helps individuals, businesses, and organizations make better decisions by providing accurate forecasts and actionable insights.

By automating complex analysis and identifying trends, the system improves planning, reduces risks, and increases efficiency. Although it has certain challenges such as data dependency, privacy

concerns, and prediction uncertainty, these issues can be managed through proper design, security measures, and continuous updates.

## Future Scope

- Integration with Artificial Intelligence
- Real-Time Data Analysis
- Mobile and Cloud-Based Platforms
- Integration with Smart Devices