Building an IPL Score Predictor - End-To-End ML Project

BEGINNER GUIDE MACHINE LEARNING PYTHON

This article was published as a part of the Data Science Blogathon

Introduction

Hello, everyone.

We know the IPL season is going on and we are all eager to know who will win the match beforehand and in the media, there is hype around the winning chances.

What if I say we can make an app that can predict the outcome, Yeah! **with the power of Machines and Deep Learning,** you can do these types of amazing stuff and this article is all about it.

In particular, here we will be looking at how you can train a model from scratch and embed it in the web app using simple and powerful libraries like **sklearn, pandas, and flask**. Also, some **web** development is involved.

The Beneath segment gives an outline and a few rules and you are prescribed to go through it previously.

Let's get started on building an IPL Score Predictor!

Overview

- 1 – Data Gathering
- 2 – EDA – Exploratory Data Analysis
- 3 – Data Cleaning
- 4- Data Preparation
- 5 – Model Development
- 7 – Conclusion
- 8 – References
- 6 – Model Deployment – Optional

We will dive into each of the sections above, which are common steps a data science team performs. This will also ensure that you learn how to approach the problem with a productive mindset.

NOTE:- It's better to create a new environment and start working there for each project as it will later help in keeping the essentials separate and increase productivity.

Legends :
H2 – Heading Main Section

H4 – Heading – Consisting of Subsections of Main
bold – Subparts Of Subsections

Data Collection for IPL Score Predictor

Before we start our project we need to gather data. This step can be done using the following 3 methods:

- Open Sources – This data is readily available in the form of structured data (rows and columns) and can be downloaded from sites like Kaggle, UCI-ML-Repository, and Open Government Data.
- Collection by Individuals – Often it happens that in some cases, data is not available so the team gathers data using tools like we scrapper or go out and gather data for themselves.
- CrowdSourcing – In this technique, people like ours help in annotating data for eg. Captcha services.

For our use case, we are going to use the IPL Scores Dataset (link in reference) which has 76104 observations and 15 features :

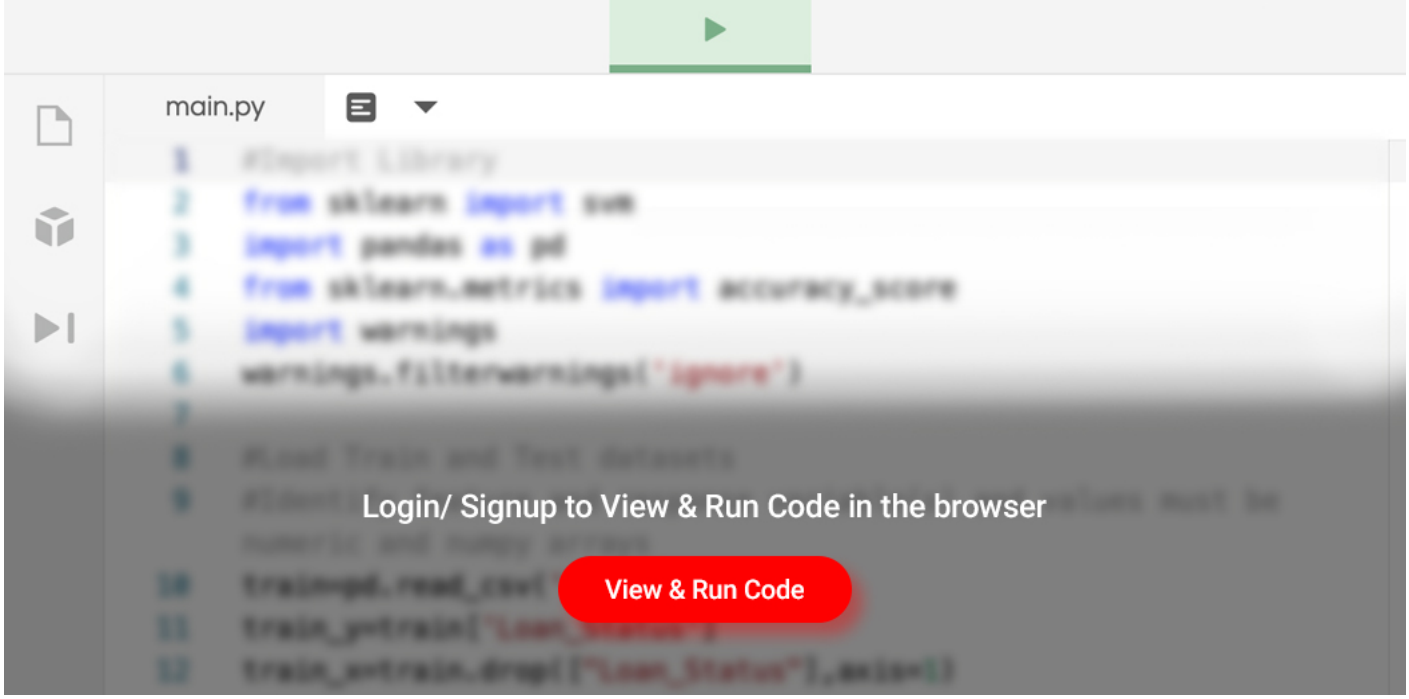
- mid - match id
- date - when matches are played
- venue - place where matches aew played
- bat_team - batting team
- bowl_team - bowling team
- batsman - batsman
- bowler - bowler
- runs - runs scored
- wickets - wickets
- overs - overs - next 3 are based on this
- run_last_5 - runs scored in last 5 overs
- wicket_last_5 - wickets in last 5 overs
- stricker - batsman playing as main 1
- non-striker - batsman playing as runner up - not main 0
- total - total score **(target variable)**

Image by Author

total – is our target variable.

Importing Necessities:

We will start by loading some essential libraries needed for the project:



The first thing to note here is that our dataset has certain columns which are irrelevant and increases resource consumption, so maybe we can remove them in future steps.

“mid,” “venue,” “batsman,” “bowler,” “striker,” and “non-striker.” # irrelevant column

EDA – Exploratory Data Analysis for IPL Score Predictor

Having looked at the data quickly, let’s dive deeper into the dataset and explore some of the insights. This procedure is very important and will allow us to understand the data and plan our next steps. Luckily pandas provide easy-to-use functions to perform our analysis. So let’s begin.

Checking Shape

Even though we know the shape of data (observations and features) the info can be wrong. So it’s better to cross-check our data once. So let’s check it.

We can get the shape of data using the *shape attribute* of pandas dataframe.

```
#checking shape of data using df.shape df.shape
```

Output:

```
>> (76014, 15)
```

Note: the output is in the format of **(rows, columns).**

Check Data Type

In real-world cases, the data we find are not of the desired data type, so it will be good if we check it too.

Pandas provide an info method that returns the data type of all columns present in the dataset.

```
# getting data type df.info()
```

Returns:

Image by Author

Notice the **type- object**, it is because panda assumes string as an object datatype.

Check Null Values

Having understood the shape and info, we will now check for null values (fields having no data – NAN's). It is essential for any project as null values can change the whole story depicted by data and can even contribute to making data worse for the use cases. Let's perform the same

All we have to do is to use the is_null method of the dataframe and then sum the outputs to get the total count for each column.

```
# check null values and then sum it up df.isnull().sum()
```

Output:

Image by Author

Seems like we are lucky as there are no null values present(0) in the dataset. and all of them are of type – int64 – takes huge memory

Check Summary Statistics

Since there is no nan value doesn't mean our data is a good representation. To get the gist, we can plot summary measures which generally include:

1. Measurement of Central Tendency – These measures allow us to understand where most of our data lies and mostly include:

- **Mean** – Average of Data
- **Median** – Center for Data
- **Frequency** – No of occurrence of specific data.
- **Mode** – Highest Observation in Data

2. Measures of Dispersion – These are the measures that allow us to understand how widespread data is and mostly includes:

- **Max & Min** – Highest and Lowest Value in the dataset
- **Range** – Highest-Lowest (captures the reach of data)
- **Variance** – Captures the variation of data (how data is varying) – Usually the sum of deviation of actual data from its mean/no of samples – 1
- **Standard Deviation** – Same as standard deviation but on the same scale as data -sqrt (variance)
- **Percentiles** – Capture the spread of data for a specific value i.e how much data is above or below it. 50% – Median

Performing each check will be cumbersome so pandas pack all these in a single function called describe. Now let's check the summary stats of our data.

```
# Check summary stats using df.describe() df.describe()
```

Output:

Image by Author

We see that for all columns the data is almost in a similar range, thus can infer that it is good data to work on.

Note: Description only works on numeric data

Check Areas For Each Numerical Feature

Now let's check how each numerical feature is contributing to our dataset and we can do this by using a panda area plot.

```
# plot area / contributions in dataset df[['date','venue', 'bat_team', 'bowl_team', 'batsman', 'bowler', 'runs', 'wickets', 'overs', 'runs_last_5', 'wickets_last_5', 'striker', 'non-striker']][:50].plot(kind = 'area', figsize = (10,10), stacked = False)
```

A Couple Of Things To Note

- **kind** – set's the plot type
- **Fig Size** – Adjust the figure size of the plot.
- **Stacked** – Insures opacity if set to False.

Returns :

Image by Author

For better illustrations, we have only used 50 observations for plotting.

Further operations can be performed. but this will suffice for now so we can move on to our next step.

Data Cleaning for IPL Score Predictor

After understanding our data, we can now proceed to clean it for our use case. We will start by dropping a few columns so that it becomes easier to work.

Removing Irrelevant Columns:

A careful inspection of our data states we have many irrelevant columns such as ‘mid’, ‘venue’, ‘batsman’, ‘bowler’, ‘striker’, ‘non-striker’.These do not contribute to data and can be removed to save memory space.

To remove the irrelevant column, all we will do is create a list of the irrelevant columns(**cols_to_remove**) and pass it to the panda’s **drop** method.

```
# Removing unwanted cols - reduce memory size cols_to_remove = ['mid' , 'venue' , 'batsman', 'bowler', 'striker', 'non-striker'] df.drop(labels=cols_to_remove , axis=1 , inplace = True)
```

***tip:* here we have used axis = 1 to drop by columns and inplace = True to ensure the operation is performed on the same dataset.**

Let’s cross-check our operations by printing out a few rows

```
#cross check df.head(3)
```

Output:

Image by Author

We can also check the shape for security purposes.

```
df.shape
```

```
>> (76014, 9)
```

We see that the columns have been dropped successfully as our column size has been reduced to 9

> This operation is irreversible and may raise an error if executed in another cell.

Filtering Consistent Teams:

Next, we will filter out teams that are consistently playing. This will allow us to have a basic set of teams that are relevant to IPL's.

1. Find Unique Teams

For that, we will first find the unique teams in bat_team, create a list out of them and then perform our filtering.

To find unique teams, we will just use the unique method over bat_team

```
# checking for how many batting teams are there df['bat_team'].unique()


>> array(['Kolkata Knight Riders', 'Chennai Super Kings', 'Rajasthan Royals', 'Mumbai Indians', 'Deccan Chargers', 'Kings XI Punjab', 'Royal Challengers Bangalore', 'Delhi Daredevils', 'Kochi Tuskers Kerala', 'Pune Warriors', 'Sunrisers Hyderabad', 'Rising Pune Supergiants', 'Gujarat Lions', 'Rising Pune Supergiant'], dtype=object)
```

Note dtype = object here is because pandas assume string data type as object data type

2. Creating List

Now we will carve out a list with consistent teams and store it in consitent_team for reference purposes.

```
# only keep current team which are present consistent_team = ['Kolkata Knight Riders', 'Chennai Super Kings', 'Rajasthan Royals', 'Mumbai Indians','Kings XI Punjab', 'Royal Challengers Bangalore', 'Delhi Daredevils','Sunrisers Hyderabad']
```

3. Filtering Consistent Teams

Finally, we can filter our required observations based on the condition

"return only those observations for which a consistent team is present in both bat_team and ball_team".

```
# filtering based on consistency df = df[(df['bat_team'].isin(consistent_team)) & (df['bowl_team'].isin(consistent_team))]
```

The above conditions are achieved by nesting the:

- **isin** method – C*hecks for required values inside columns or rows of the data frame* and returns the observations that pass.
- & operator- Ensures only to return values when both conditions are true.

4. Checking Results

Now, let's check the results by printing the unique values again which should return the same unique values.

```
# printing out unique team after filtering print(df['bat_team'].unique()) print(df['bowl_team'].unique())
```

Image by Author

Hurray! As expected, we have now filtered the consistent teams. Off to the next process.

Filtering Based On 5 Overs

HYPOTHESIS:- We can assume that "*In most of the matches the actual game starts after elapsing 5 overs",* so it can serve as a good starting point for our training data.

So following our hypothesis. We will just return all the observations after 5 overs by first accessing the overs column of the dataframe(pdf) and using >= operator.

since for every match one can predict more accurately if one has 5 over data so,
filtering based on 5 overs
df = df[df['overs']>=5.0]
df.head()

Returns:

Image by Author

We can see that the over columns have only values > 5.

Using these types of hypotheses is very common and can even yield good results as long as it satisfies in real-world, domain knowledge necessary though

Change Date Column Type

While examining the info on the features column, we found that the data column is an object data type on which no operations can be performed.

We will use the date-time library to convert the data column into a date-time object.

```
# converting date cols from string to date time object from datetime import datetime df['date'] = df['date'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d'))
```

Understanding Code:

- **df['date']** : Accessing Date Column'

- **apply**: Applies a certain operation over the entire dataframe
- **lambda**: One line in having syntax [lambda iterator : expression]
- **expression**: strip the values of the date-time column as YYYY-MM-DD(as was present in the column)
- **(%Y-%m-%d)** : format specifier for *strptime(x,format)*.
- We are using '.' to chain multiple methods together and make our code short.

Now let's cross-check:

```
df['date'].dtype
```

```
>> dtype('<M8[ns]')
```

So it seems like we have successfully converted the date column to date-time data_type

> This operation is irreversible and may raise an error if executed again.

There are further steps involved but this will suffice for our use case.

Data Preparation for IPL Score Predictor

Now, as evident from the info, our data takes a variability of data types including strings, data time, and numbers. But our model requires them all to be in a numeric format, so let's see if we can perform some operations to make it model-friendly.

Encoding Categorical Variables:

On careful inspection, we can find that bat_team and bowl_team are categorical data and can be encoded as numbers(0/1). This is called ONE HOT ENCODING and can be achieved by pandas get_dummy function.

```
# converting categorical features using 'One Hot Encoding' # for numerical values cat_df = pd.get_dummies(data = df, columns = ['bat_team' , 'bowl_team'])
```

Understanding Code

- **cat_df** – New dataframe having categorical data encoded with each category as a separate column (refer to the output).
- **data** – original data frame(df).
- **columns** – Columns having categorical data in this case 'bat_team' and 'bowl_team'.

Now let's see what fun has to offer in return:

```
cat_df.head(2)
```

Output:

One Hot Encoding Results – Partial View – Image By Author

As can be seen, the columns have been increased and one-hot encoded. While it is taking up more space, that's a trade-off to consider while encoding is up to individuals.

Another method is to use a dictionary(key: value) mapping which can be used with the "apply" method.

It's now time to split our data into (training and testing set) so that we can fit it into our model.

Splitting Dataset:

As a general case, we split our data according to ratio train = 80% and test = 20% but here we will be learning how to adapt for a split for a problem domain.

What we will do is, instead of going for split size based on ratio, we will perform the split based on years.

```
# split the data into train and test set - based on date column X_train = cat_df.drop(labels = 'total', axis = 1)[cat_df['date'].dt.year <= 2016] X_test = cat_df.drop(labels = 'total', axis = 1) [cat_df['date'].dt.year >= 2017]
```

So all I did here is define the variables, drop the last column, and filter out the observation based on conditions that return only year values from the date column.

For labels we only use a single column (total) which Panda assumes to be a series so we need to use values over it, else it will return an error sometimes.

```
# since only one column so cosidered as series y_train = cat_df[cat_df['date'].dt.year <= 2016]['total'].values y_test = cat_df[cat_df['date'].dt.year >= 2017]['total'].values
```

Theory:

- **X_train** = All training features except the target column(runs) up to the year 2016
- **X_test** = All training features except the target column(runs) for the remaining years
- **y_train** = target column up to the year 2016
- **y_test** = target column for the remaining years.
- (X_train, y_train) – Training Set & (X_test, y_test)- Testing Set

Let's check the split now for cross-verifying.

#checking shape
print(X_train.shape , y_train.shape)
print(X_test.shape , y_test.shape)

```
>> 37330, 22) (37330,) (2778, 22) (2778,)
```

The split was successful. Also notice the (**2778**,) for y_test is because it is a single column with 2778 observations.

Dropping Date Column:

One last thing we can do before closing this section is to drop off the data column from our training and test sets, as it is of no use to us now as this will help free up some memory space for further operations.

We will use the same drop method used earlier to perform the dropping.

```
# since the requirement of our date column is over so we can drop it # dropping date column X_train.drop(labels = 'date', axis = True, inplace = True) X_test.drop(labels = 'date', axis = True, inplace = True)
```

Now let's check how splits look using the "display" method – Jupyter specific.

```
# use display to cross check in single line - only one display("X_train", X_train.head(1)) display("X_test", X_test.head(1))
```

Output:

Image by Author

Our data looks pretty and as expected for model feeding (all numeric values).

Model Development for IPL Score Predictor

If you have come this far with me – congrats, you have understood the essence of being a data scientist (yep that's what a data scientist does!).

Now the only thing left is to choose a model/build it for training, evaluate the results, and finally save it for the latter use case.

Model Selection & Training:

After all the hard work we did, it's now time to choose a model architecture for our use case. This will be a function that will find a way to map our training set to training labels thus allowing prediction of the score and given input data.

Since we have seen that the data mostly have a linear relationship (as evident from the description) we will use a simple linear regression model for our use case from sklearn.

1. Importing Model

Before any further steps, we will first import the **Linear Regression** model from *sklearn's* linear_model class and instantiate a linear regression object as a reg.

```
# import module
from sklearn.linear_model import LinearRegression
reg = LinearRegression() #init
```

The job of the linear regression model is to find the line of best fit, for which error/loss is very low

2. Training Model

Now it's finally time to train our model. To do so we just use the fit method and pass our training set.

```
# training model reg.fit(X_train , y_train)


>> LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

That's pretty much it, we have trained our model and can be used to predict our test set.

For more details on Linear Regression, consider checking the official documentation page.

Model Evaluation

After training comes the evaluation part which tells how our model is performing on data other than the train set. If it performs well, it is good to be deployed in the wild, else you need to reiterate the entire process(or at least a few processes, boils down to how you apply logic).

Let's test our model too using the prediction method. We will be passing the X_test to the same.

```
# getting predictions prediction = reg.predict(X_test)
```

The results returned are predictions and are not user-friendly to understand, so it is better to plot them.

1. Visualizing Results

Since we have predictions and actual data(y_test) we can use Seaborn's distplot which will plot the *distribution of prediction vs actual data* – in simple words difference between 2(as distance measure).

```
# plotting our fit import seaborn as sns sns.distplot(y_test-prediction)
```

Returns:

Image by Author

We can notice that the difference between **actual** and **predicted** data (line) is not much and resembles a Gaussian/normal distribution(data symmetric around the mean) as evident from the bell curve.

To further check this, we can also use the evaluation metrics

2. Using Evaluation Metrics

Plots are a great way to visualize results but it will be much more convenient if we can just summarize the result in a single metric.

Luckily we have a few stats metrics provided by sklearn itself for the task. These include MAE, MSE, and RMSE.

```
# checking for scores from sklearn import metrics
```

We just imported metrics from sklearn

```
# checking for scores from sklearn import metrics import numpy as np # Mean Absolute Error print('MAE: ', metrics.mean_absolute_error(y_test , prediction)) # Mean Squared Error print('MSE: ', metrics.mean_squared_error(y_test, prediction)) # Root Mean Squared Error print('RMSE: ', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

Returns:

Image by Author

Great, we have our **MAE** to be *12 &* **RMSE** to be *15*, we can now proceed further as our score matches the plot. However, if you like you can tweak the data and model to get a score even lower than this (quite possibly!).

NOTE: MSE – Is large because of the square terms in the formula (refer to figure below)

Evaluation Metrics

Finally, we can now save our model.

Saving the IPL Score Predictor Model

To save our model we will be using the pickle library and dumping our model(refer here to learn more)

Here (dump) means to parse the value to create a serialized object using a pickle.

```
# creating our model pickel file - saving model file_name = 'ipl_score_predict_model.pkl' pickle.dump(reg , open(file_name,'wb'))
```

Here ‘wb’ refers to *write binary* and open is a constructor which creates a file provided in the file_name(File I/O).

This will return the model file in the directory.

Output:

Image by Author

Now let's see how to deploy our model in the next section.

Model Deployment – OPTIONAL

The next step is to deploy our model but before that let's choose the deployment architecture. Here is a quick summary image to get started with.

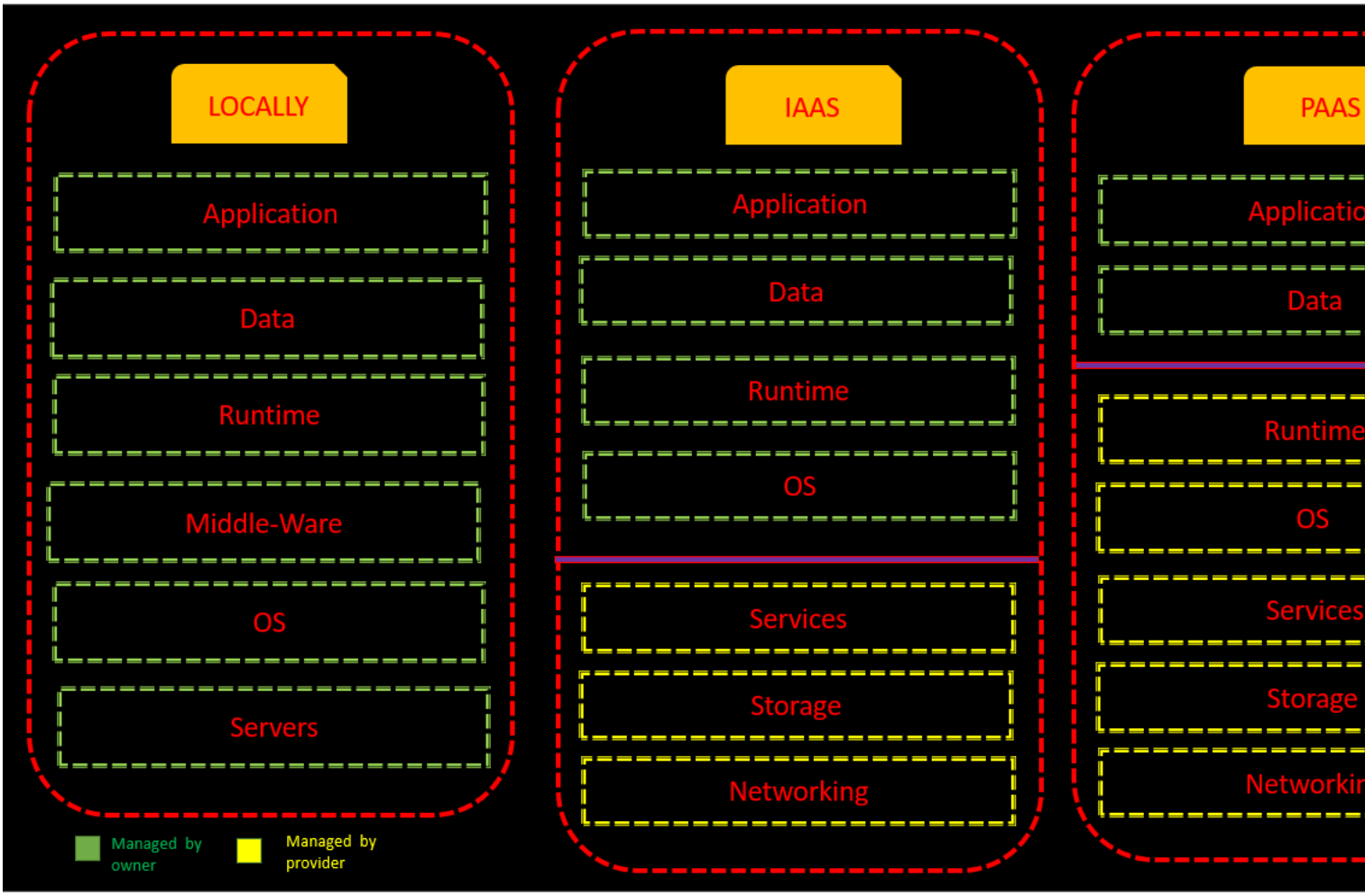


Image by Author

PAAS – Platform As A Service architecture is convenient as we only need to take care of our data part. So we will choose it.

For the platform, we will be using Heroku.

Deployment Using Heruko

For deployment, one can get lost easily, so we will follow the below steps in sequence

1. Signup On Heroku
2. Create A Web App Using Flask
3. Commint Code In Github
3. Link the Github To Heroku
4. Deploy The Model
5. Visit The Webapp

Image by Author

1. Signup

We will just go to Heroku's official site and **signup**. Making sure we select the role as a student will allow us to get free dynos.

Finally, we will download the CLI tool provided as it will allow us to see the logs.

2. Creating a Web App Using Flask

Next, we will create a simple web app using **HTML**, **CSS**, and **JS** for the front end & **flask** and **request** for the backend.

We will create the following files:

- **index.html**: This will be our home page and will be rendered when someone accesses the app. Here we will take user input using the from the user too.
- **result.html**: This is the page that will render the output after getting prediction in the backend.
- **app.py**: The main app that will be responsible for handling all backend operations (by routing requests using methods like '**/**' and '/**predict**'. This will also include a one-hot encoding technique that we use during training to ensure there are no errors while using data.
- **Requirement.txt**: File which lets the platform know which libraries need to be installed during deployment.Can be created using:

```
pip freeze > requirements.txt.
```

- **Procfile:** Essential file to let the platform know what app to run when deployment is done. **gunicorn** is required*

This is what our working directory should look like:

Image by Author

Image by Author

Note:

- **Static and Template**: Folders are for storing resources and webpages respectively. Considered good practice.
- **app: value** – Value should be app name defined when initializing flask app – line 10.

3. Commit Code on Github

Now let's head over to git and create a new repository called ipl-demo. Make sure to set it to the public else the deployment will not happen.

Image by Author

Next, we will commit our code and some other files. Folders need to be dragged and dropped manually.

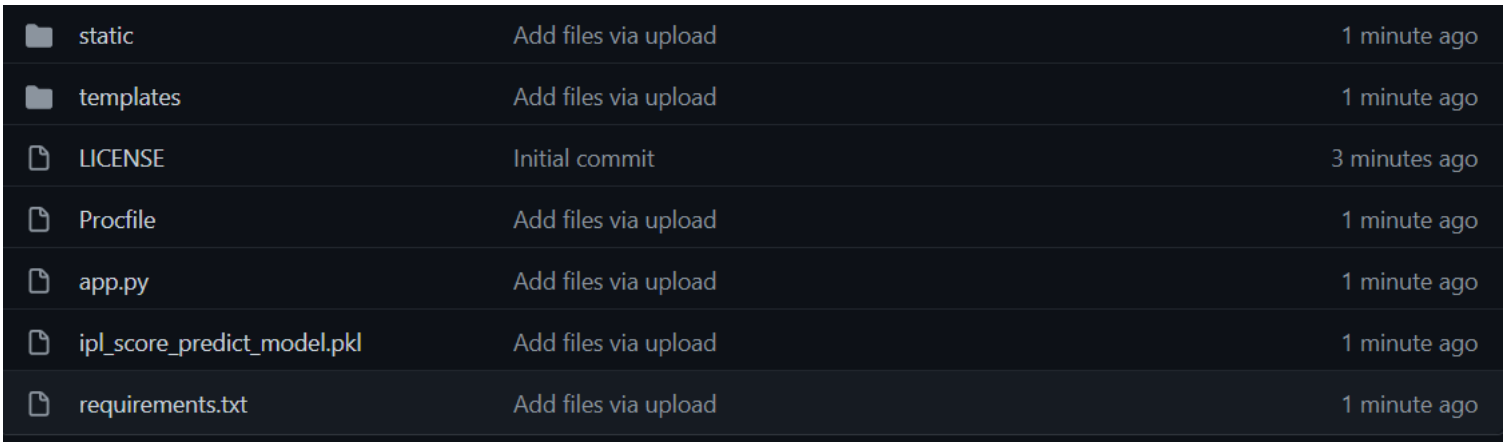


Image by Author

That's all our work is done here. So let's move on to the next part.

4. Link Github to Heroku

To let Heroku take care of every step, we need to connect to our GitHub account.

1- You can do so by logging into the created account on the platform and following the instructions below:

1- On the home page, click the new button.

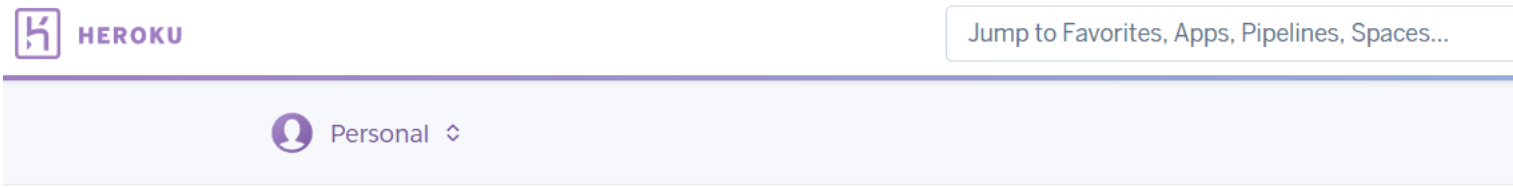


Image by Author

2- Fill in the details. we need to add a unique app name – ipl-score-predictor (in our case).

Image by Author

3- Connect your GitHub account. While searching, we provided the same repository name we created on GitHub (imp) and clicked connect.

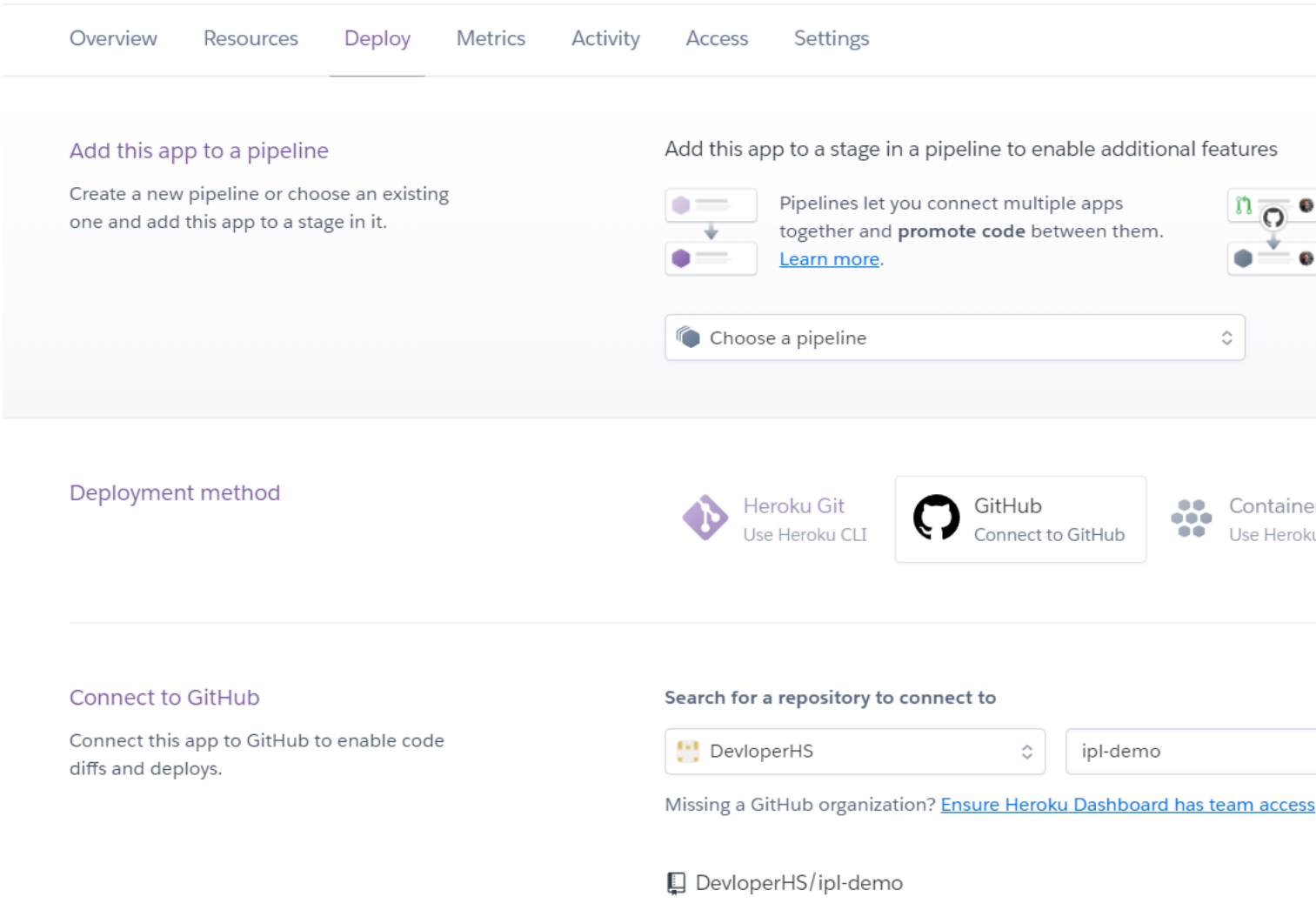


Image by Author

5. Deploy the Model/WebApp

To deploy, we can use either automatic deployment or manual deployment. To keep things simple, we will just click the Deploy Branch and then wait for a minute or 3.

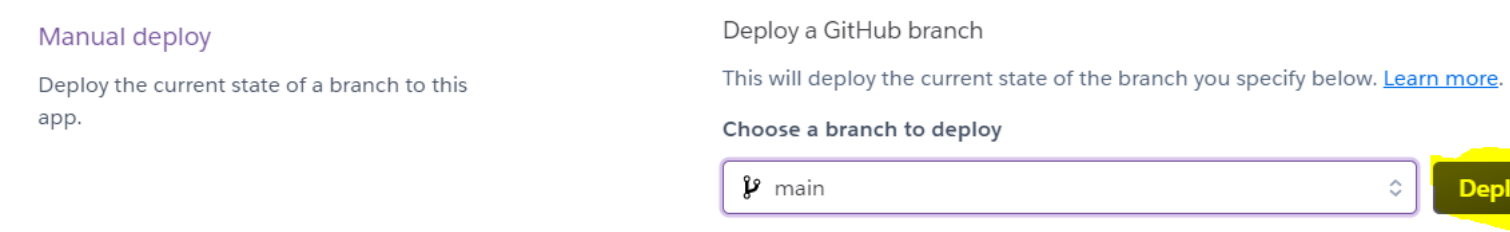


Image by Author

OUTPUT:

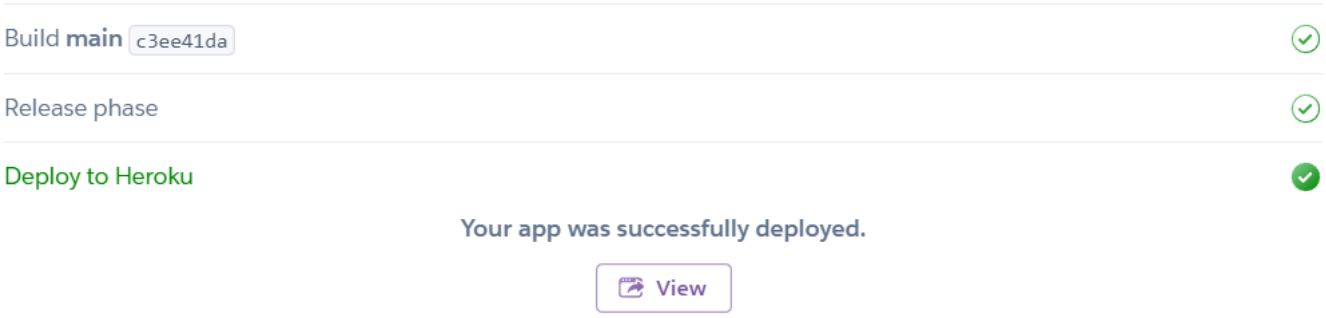


Image by Author

NOTE: The use of automatic deployment is preferred as it makes changes automatically as we upload new model versions

5. Visit the Webapp

Looks like we have successfully deployed our model. To access the app, all we need to do is to click on the View button shown in the image above and test our web app.

Here is a glimpse of what we have achieved:

First Innings Score Predictor for *Indian Pre*

A IPL Score Prediction WebApp

Mumbai Indians

Mumbai Indians

7.2

64

4

34

45

Predict Score

Image by Author – Input (index.html)

First Innings Score Predictor for *Indian Pre*

A Machine Learning Web App, Built with Flask, Deployed usin

The final predicted score (range): -104 - -

Image By Author – Output(result.html)

We can now share this with anyone in the world using the site URL.

CONCLUSION

Here are a few of the conclusions we can derive from this article:

- ***Reality*** – As thought, data scientists tend to work more on the data cleaning and processing part(about 80-85%) rather than making cool predictions and insights.

- **Scope for Improvement** – There is a lot of room for improvement of our model, for example, we haven't taken venue into account but it can be taken, or we can try different model art architecture to fix the curve fit issue at the center(can use polynomial fit), or we can even try changing the data and the split size. It is all up to individual imagination.

-

- **Deployment** – Deploying models is a good way to showcase to the world what one is capable of along with fixing real-life issues.

- **Work Process** – As can be seen, following steps in a procedural manner can simplify our problem solving and is generally preferred in the industry.

- **Environment** – As we also saw earlier, it would be good to keep important files separate, so prefer creating one.

With this, we have come to an end with this daunting article/guide. **This has not been an easy one to write, but I hope it will be an easy one to understand and you will apply the knowledge you have gained through this guide.**

Feel free to comment on any suggestions in the comments section below . Also, if you want to connect with me, reach me on LinkedIn / Github / Twitter.

References

Here are a few of the references who are eager to learn more:

Image Ref: Evaluation Metrics

Inspiration: Inspiration For Learning

Code and Related Files: Github Repository

Thanks For Reading😊

Article Url - https://www.analyticsvidhya.com/blog/2021/10/building-an-ipl-score-predictor-end-to-end-ml-project/

Purnendu Shukla

Hey All,

My name is Purnendu Shukla a.k.a Harsh. I am a passionate individual who likes exploring & learning new technologies, creating real-life projects, and returning to the community as blogs.

My Blogs range from various topics, including Data Science, Machine Learning, Deep Learning, Optimization Problems, Excel and Python Guides, MLOps, Cloud Technologies, Crypto Mining, Quantum Computing.