2018

Horse Racing predictive analysis

Capstone Project By:

Tanmay Uttam Sandav (182001274)

Under guidance of:

Dr. Sergei Schreider

Assistant Professor of Professional Practice.

R BUSINESS

Table of Contents

1. Abstract

Horse Racing is an equestrian performance sport, typically involving two or more horses ridden by jockeys over a set for competition. Estimating horse racing result has been a popular topic in machine learning field, whilst the possibility of profit earning is depending on the accuracy of predicting the probabilities of horses to win in a race[1]. Horse Racing Predictive Analysis project undergoes a standard data analysis and modelling process. One interesting fact has been found, such that public intelligence is performing not bad in horse racing, thus the objective of the project is to develop a model which could perform as good as or even outperform the public intelligence. This report would discuss the method of feature selection and normalization, the reason of proposing new features, the possible ways to train the model, the difficulties when handling unbalanced dataset, the method to evaluate the model and the results derived from different learning algorithms. We show that it is possible to construct a model outperforms the public intelligence, also by setting some threshold and not participating in every race, it is possible to generate profit through the model trained with deep neural network and the model driven by pattern matching. Data structure and all requirements are compiled by Indian Horse Racing website thus data will be web scrapped from indiarace.com. The cleaned and processed data will undergo data modelling of linear models and deep neural network models like Artificial Neural Networks. Obtained results will be analyzed and compared with real-time Indian horse racing results. Selected model will be optimized based on the results. Python 3.0 will be used for programming and data will be stored in spreadsheets.

Figure.1 Horse Racing in India

[1] https://en.wikipedia.org/wiki/Horse_racing

2. Introduction

Motivation

Ever since I was a teenager I have enjoyed gambling and Horse Racing in particular. I grew up in Mumbai, India home to one of the most renowned race tracks in the country. The excitement is palpable at the track, with tourists, hardened gamblers, and fans of the sport all cheering on their selections as they come down the stretch. I am a fan of the sport in two capacities. Firstly, I'm a fan as anyone else is a fan of baseball or football. Secondly, I am a fan as a gambler. Horseracing and Indian Poker are the two main games that are mathematically possible to win at. The great horses of my era were never great investments, but they have memorable wins that elicit nostalgia and excitement. They represent the dream that if your skill eclipses the house edge, then you're not simply throwing your money away, but are potentially making money. Of course, it's no small feat to beat the house edge in either game, but for the past few years, it's exactly what I've been trying to do. Also, Horse racing has been a famous topic in machine learning field, while the recent performance of deep neural network is stunning and there were a lot of new machine learning tools released recently, which could let us apply deep learning algorithm or other machine learning algorithm easily, so that we would like to conduct an experiment on predicting horse racing result.

Background

Horse Racing

Horse Racing is a sport which involves two or more horses ridden by Jockeys and trained by Trainers running from a start point to finish post. The distance is different for different races.[2] Winning stakes differ based on the category of horses running, distance of the race and various other factors. Horses Races are conducted throughout the year at different centers but are telecasted at all other centers at real time. Horse Racing include several skilled employees. Example, a horse participating in a race needs a jockey who rides the horse, trainer who trains the horse for years, workers in the stable who feed and train the horse timely under supervision. Also, there are several other people working behind the scene. Every horse is owned by either an individual or group of individuals. At last but not the least are the bookmakers and punters. A bookmaker is the one who puts the odds and accepts bets from punters. Odds are mathematically calculated depending on the horse's previous runs, class, age, pedigree, track records, backings from punters and a few other factors. Punters bet on the horse based on its odds and intangible elements like fear, greed and maybe some insider knowledge of how the horse may perform on that day.

Indian Horse Racing

Horse racing in India is over 240 years old. The first racecourse in the country was set up in Madras in 1777. Today, India has a very well-established racing and breeding industry, and the sport is conducted on nine racetracks by six racing authorities. Racing is restricted to Indian-bred racehorses and India has a well-established breeding industry with stallions imported from all over the world. The Indian Stud Book maintains records of all thoroughbred breeding activity in India.[3] India has a mixture of both Tote betting and traditional bookmakers. The sport has glamour and recognition attached to it. Many of the horses are owned by renowned personalities who pursue this as a stylish and classy sport.

[2] https://en.wikipedia.org/wiki/Horse_racing
[3] https://en.wikipedia.org/wiki/Horse_racing_in_India

Indian Horse Racing is hosted by 7 turf clubs at 9 centers[4]. The information about turf clubs, centers, and their seasons is as follows:

- Bangalore Turf Club, conducts racing at Bengaluru in two distinct seasons - in summer from May to August and in winter from November to April.
- Hyderabad Turf Club, conducts racing in Hyderabad where racing is held on the Monsoon Track from July until the end of October and on the Winter Track from November until February. Hyderabad usually races on Sundays and Mondays.
- Royal Calcutta Turf Club, conducts racing in Kolkata, with a main winter season from November to April and a monsoon season which runs from July until mid-October.
- Royal Western India Turf Club (RWITC), conducts racing in Mumbai from November to May and Pune from July to November.
- The Mysore Race Club conducts racing in Mysuru, is the most picturesque in the country. Set up in the foothills of the imposing Chamundi Hills, the Mysore Race Club has a distinct identity of its own. Mysore, which is an independent Turf Authority conducts regular season between mid-August and the end of October, as well as smaller summer and winter seasons.
- Madras Race Club, conducts racing in Chennai, home turf of the late turf baron Dr MAM Ramaswamy.
- Delhi Turf Club, established in 1940, which conducts racing at India's capital Delhi usually once a week from August until May, where racing is run under the aegis of RWITC.

[4] https://en.wikipedia.org/wiki/Horse_racing_in_India

Major Races

India has five 'Classic' races which parallel the original British classic races. The Indian 1,000 and 2,000 Guineas are run in December. The Indian Oaks is run at the end of January. The Indian Derby is run on the first Sunday of February and carries a purse of over ₹ 30,000,000. Lastly, the Indian St. Leger is run in September. They are all run in Mumbai, apart from the St. Leger which is run at Pune.[5]

The Invitation Weekend which rotates between the various turf authorities is held on the first weekend of March. This features a Group 1 race each for sprinters over 1200 meters, a race over a mile and a 3000-metre race for stayers. The best horses are invited from all over the country for these races. The showpiece event is open to Indian horses which are 4 years old and over, invited from all the turf authorities, and carries a winner's prize of ₹ 10,000,000. The Bangalore Derby is held on the second Sunday of July in Bangalore every year. It is sponsored by Kingfisher. Invitation cup and associated races sprinter, stayer, Super mile is rotational, Hyderabad 2014, Calcutta 2013 Bangalore 2012, Bombay 2011 and this year is RWITC' Mumbai's Turn, is run over 2400 meters was for only 4 years old only were eligible but from 2014 onwards it has been changed to elder horses also.

[5] https://en.wikipedia.org/wiki/Horse_racing_in_India

Betting Guide

How to bet: a beginner's guide[6]

You can bet either on the Totes or Totalizators or in the bookmakers' ring. One prerequisite of betting is that you must be 18 years or older.

Betting on the Totes

You can go to a manned Tote window, buy a cash voucher from a Tote Service Outlet, or call the roving operator with a hand-held computer. You can either bet at variable Tote odds or at fixed Tote odds. With the fixed Tote odds, your winning amount are at the odds prevailing at the time you make your wager are guaranteed and printed on the ticket.

Win (Minimum bet: Rs.10/- only): Pick a horse to finish 1st.

Place (Minimum bet: Rs.10/- only): Pick a horse to finish 1st, 2nd or 3rd (in races with 8 runners) or 1st, 2nd, 3rd or 4th (in races with 12 or more runners). With 7 or fewer runners, your choice must finish 1st or 2nd.

Accumulator Win (Minimum bet: Rs.10/- only): This is a multi-race bet in which the winnings are subsequently wagered on each succeeding nominated race.

Kenchi: A permutation of the Accumulator Win or Place wager accepted for a chosen group of 3, 4 or 5 races. The good news with Kenchi is that the chances of your getting back some money are better than with a straight Accumulator. For the minimum unit of Rs.10/-, a 3-race Kenchi requires you to invest Rs.40/-; a 4-race Kenchi, Rs.110/-; and a 5-race Kenchi, Rs.260/-.

Forecast Pool (Minimum bet: Rs.10/- only): Pick two horses to finish 1st and 2nd.

Quinella Pool (Minimum bet: Rs.10/- only): Just like the Forecast Pool except you win if your choices finish 1st and 2nd or 2nd and 1st. In other words,

[6] http://www.rwitc.com/bettingentertainment/beginnersGuide.php

so long both your chosen horses figure in the first two, the order of finish does not matter.

Tanala Pool (Minimum bet: Rs.10/- only): Pick 3 horses to finish 1st, 2nd or 3rd in the exact order or buy a combination ticket where your first three choices finish in the exact 1-2-3 order to earn the 70% dividend. The consolation 30% dividend is paid on the ticket where the order for the second and third place gets interchanged; in other words, 1-3-2 instead of 1-2-3.

Jackpot (Minimum bet: Rs.10/- only): Select the winners of the five races designated by the Club for the Jackpot Pool at the declaration stage. You can either buy a single-digit ticket (e.g., 1-8-5-2-12) or a combination ticket (e.g., 1/6-7/8-5-2-6/12). If you get the first four winners correct, you share in the consolation dividend of 30%. If you get all 5 legs right, you share in the 70% dividend amount.

Super Jackpot (Minimum bet: Rs.5/- only): RWITC has a Super Jackpot pool with six legs or six races designated for the Super Jackpot Pool at the declaration stage. You can buy either a combination or a single ticket. If you nominate the winners of the first five legs, you share in the 30% dividend amount. If you get all 6 legs right, you share in the 70% dividend amount.

How to Bet With A Bookmaker:

Win (Minimum bet: Rs.100/- plus taxes): Pick a horse to finish 1st.

Place (Minimum bet: Rs.100/- plus taxes): Pick a horse to place.

3. Objective:

Our objective is to create a model that which could predict winner and perform as well as public intelligence or even beat public intelligence in terms of accuracy. The target is approached in two parts, firstly I predict from a race card if any of the horse can win and secondly what would be the classification of all the horses in places. Machine Learning algorithms like Logistic Regression, Decision Tree classifier, Support Vector Machine classifier, Ensemble methods like Random Forest classifier and Artificial Neural networks have been used to train the data and predict the results.

4. Data Collection:

Fast Approach

I could have contacted the web developers and support staff and asked for providing me the data desired for analysis. The reply could have been positive or negative and the data obtained wouldn't have been in the desired format. So, I decided to use an alternate and reliable approach of Web Scrapping.

Web Scrapping

Tailor-made python scripts were created to scrap data from the www.indiarace.com website, historical data and horses' information for the Mumbai Season 14-15, 15-16 and 16-17 seasons was collected. Data was stored in a pandas data frame and later exported into a csv file. The first challenge was finding data. The entities that own the data control it tightly. Results from individual races are findable on the internet, but there is not a single location that has all the data, nor is there a compiled database that was available to us. My first challenge, therefore, became creating a database upon which we could train models. Luckily, the data available on the website was in a tabular form which eased web scrapping. I used beautiful soup in Python to scrape over 3 web pages of data. I was able to build an initial database with records from the 1 major tracks in India, spanning almost 3 years of data.

- Track: 1 Track with 3 years of races
- Horses: Over 1550 individual horses, running over 9030+ times
- Jockeys: More than 120 jockeys, some with horses running over 956 times
- Trainers: 121 trainers, some with horses running over 956 times
- Races: Over 950 races captured

The database was formatted as a single row for each horse appearing in each race, totaling 9032 running's. The data in each record included information about the horse and its lineage, the trainer and jockey, and statistics including the horse weight, speed rating, race-time odds, gate draw and age, as well as the variables we might want to predict, the finish place and finish time.

5. Data Storage

Tools

Pandas Data Frame

Web scrapped data from www.indiarace.com was stored in pandas data frame initially. Columns for the data frame were specified first and an empty data frame was created. Tabular columns were mapped to data frames columns and data was appended race wise. Data captured in data frame was then exported to a csv file.

```
df = pd.DataFrame(columns = ['Pl No', 'H.No','Horse','Desc','Trainer','Jockey','Wt','Al','Dr','Sh','Won By','Dist Win','Ratings',

for i in dates:
    url = 'http://www.indiarace.com/Home/racingCenterEvent?venueId=2&event_date=' + str(i) + '&race_type=RESULTS'
    pages.append(url)
```

```
for item in pages:
    page = requests.get(item)
    soup = BeautifulSoup(page.text, 'html.parser')
    race_1 = soup.find_all(class_='table-responsive table-border-radius')


    for i in race_1:
        race1=i.find_all(class_='dividend_tr')
        for row in race1:
            cells = row.find_all("td")
            links = row.find('a')
            pl_no = cells[0].get_text().strip('\n')
            hno= cells[1].get_text().strip('\n')
            horse=links.contents[0].strip('\n')
            age=cells[3].get_text().strip('\n')
            trainer=cells[4].get_text().strip('\n')
            jockey=cells[5].get_text().strip('\n')
            wt=cells[6].get_text().strip('\n')
            al=cells[7].get_text().strip('\n')
            dr=cells[8].get_text().strip('\n')
            sh=cells[9].get_text().strip('\n')
            wb=cells[10].get_text().strip('\n')
            dw=cells[11].get_text().strip('\n')
            ratings=cells[12].get_text().strip('\n')
            odds=cells[13].get_text().strip('\n')
            time=cells[14].get_text().strip('\n')

            df=df.append({'Pl No':pl_no, 'H.No':hno,'Horse':horse,'Desc':age,'Trainer':trainer,'Jockey':jockey,'Wt':wt,'Al':al,'D

df.to_csv('Mumbai15-16-17.csv')
```

PostgreSQL

Figure 2. PostegreSQL Logo

PostgreSQL is an open source Relational Database Management System. The reason we choose to use it is because it has a good OSX GUI client, and SQL is good for extracting data from database. PostgreSQL was used in little capacity for data exploration and to impute missing values.[7]

MS – Excel

MS – Excel is a database kind of application for cross platform operating systems like Windows, macOS, Android and iOS. The features like calculation, graphing tools, pivot tables and a macro programming language called Visual Basics.

A dataset of 9032 rows and 15 columns is stored in the spreadsheet. The columns are Pl No, H No, Horse, Desc, Trainer, Jockey, Wt, Al, Dr, Sh, Won By, Dist Win, Ratings, Odds, Time. The spreadsheet head is as below:

Pl No	H.No	Horse	Desc	Trainer	Jockey	Wt	Al	Dr	Sh	Won By	Dist Win	Ratings	Odds	Time
1	7	GYAHANA	4y b m	D J Surti	Vishal	51.5	-5	4	S	NECK		6	25/1	01:12.7
2	1	MOROCCA	4y b g	M Bobby	R Ajinkya	61.5	-3.5	5	A	1/2		26	72/100	01:12.8
3	4	DEVOTED	3y b g	Ivor Ferna	R Shelar	58.5	-	8	S	1/1	2	20	13/1	01:13.1
4	8	PRACS	4y b m	Nirad Kara	S Zervan	50	-	2	A	1/8	3.5	3	14/1	01:13.3
5	3	ZAFIRAH	5y b m	Narendra I	Dashrath S	59	-	3	A	1/2	11.5	21	27/4	01:14.6

[7] https://www.postgresql.org/

MS – Excel provides varying range of formulas and analysis tools for visualization and data manipulation. But I chose to use python since I am comfortable coding in programming languages. Also, python provides more libraries for analysis, data cleaning, data exploration, data modelling and model training. So, I used MS – Excel in more capacity for data storage rather than manipulation.

6. Dataset details and Data Exploration:

Pl No - The Pl No column specifies the race results. 1 indicates winner for that race card, 2 indicates runner up, 3 indicates third place and so on. Pl No is a categorical variable containing numeric values that don't mathematical meaning. The Pl No is a finite value taking a range from number of horses in the race card. The only other possible value that can be taken by the Pl No column is 'W'. W means that the horse was withdrawn before the start of the race due to some reasons and circumstances. So, in that case, if there are 13 horses in the race card and one withdraws from the race than the rest horses get numbers according to their finishing at the post. Following is the grouping of values in the Pl No column:

```
In [8]: df['Pl No'].describe()

Out[8]: count     9032
        unique      22
        top          1
        freq       956
        Name: Pl No, dtype: object

In [9]: df.groupby('Pl No').size()

Out[9]: Pl No
        1     956
        10    442
        11    330
        12    233
        13    161
        14    105
        15     56
        16     32
        17     22
        18      9
        19      3
        2     955
        20      2
        21      1
        3     948
        4     927
        5     890
        6     811
        7     754
        8     661
        9     557
        W     177
        dtype: int64
```

Horse No - Horse No is a column containing discrete values. The Horses are numbered on their participation in that race. Horse No is a value ranging from one to the number of horses in the race. It is easy to recognize a horse from the race while it races rather than remembering it based on body structure, color and other things.

Horse Name: Horse Name is the categorical variable stating the horse name in the race card. Horse Name is repetitive in different race cards but not in the same race card. There are 1560 unique horses in the dataset. Horse Name being a categorical variable need to be label encoded so that the text data converts to numeric and then the machine learning algorithm can understand those values. Horse Name based on its placing can be used for data analysis and exploration. Following is the distribution of horse participation over the past 3 years.

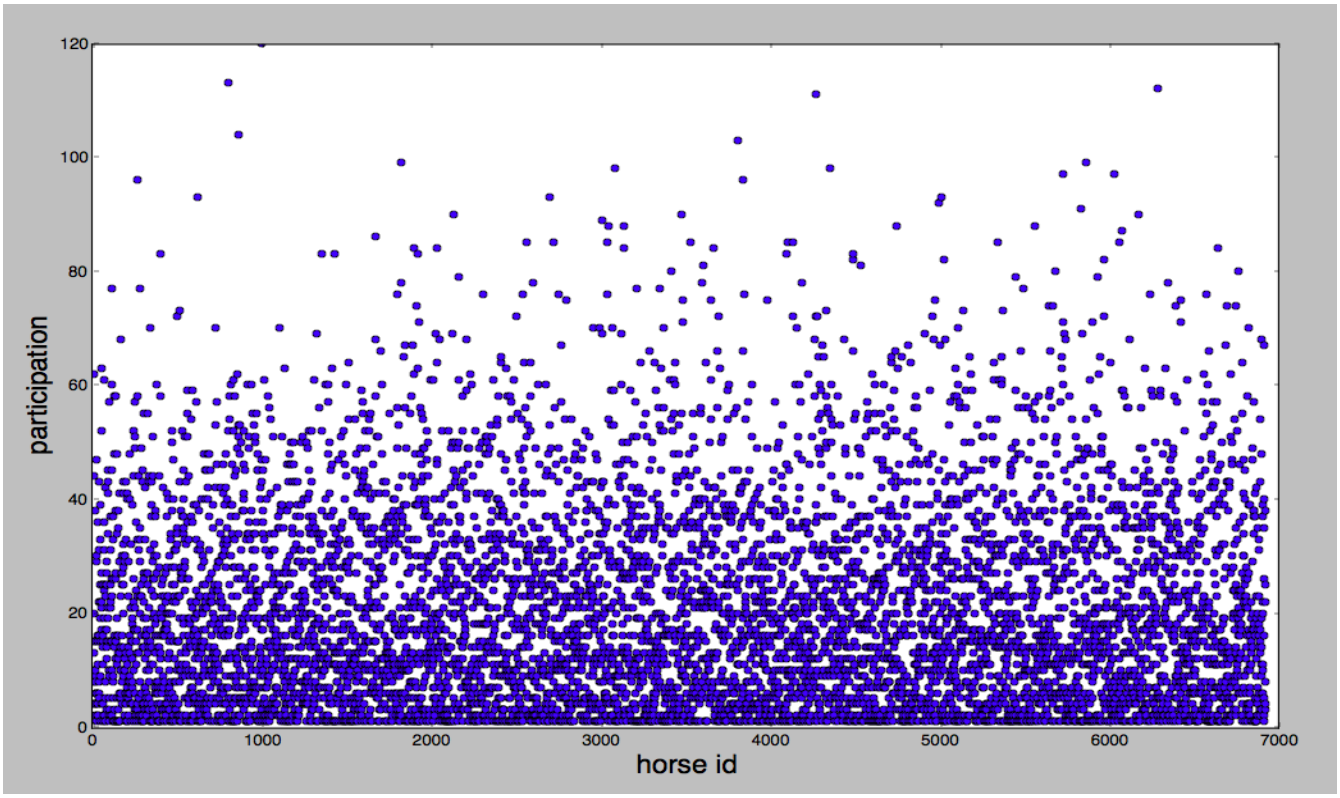

Figure 3. Horse Distribution

DESC – Desc donates the age and the sex of the horse. This is a categorical variable having importance in model training. Examples of data in this column is 4y b m. 4y indicates the age.

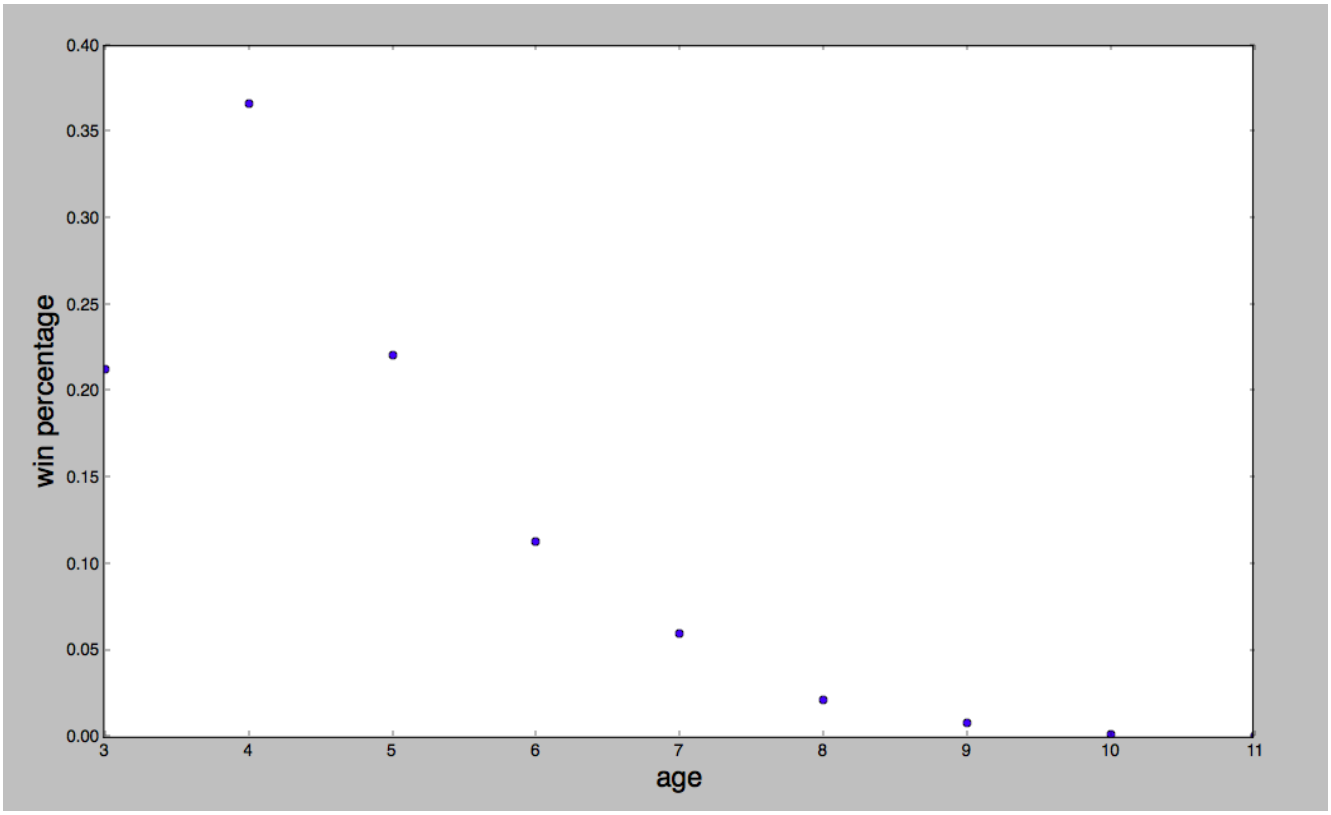

Figure 4. Age Distribution

Trainer Name – Trainer name is the categorical variable stating the name of the person under whose name the horse gets trained. Trainer name can repeat in same race and in different races as well. A trainer can have couple of their horses running in the same race. Trainer name column contains null and unmeaningful values which need to be imputed or modified before model training. Trainer name being a categorical variable need to be label encoded so that the text data converts to numeric and can be used in training machine learning models. There are 132 unique trainers in the dataset. This variable is significantly important in predicting the results of the race.

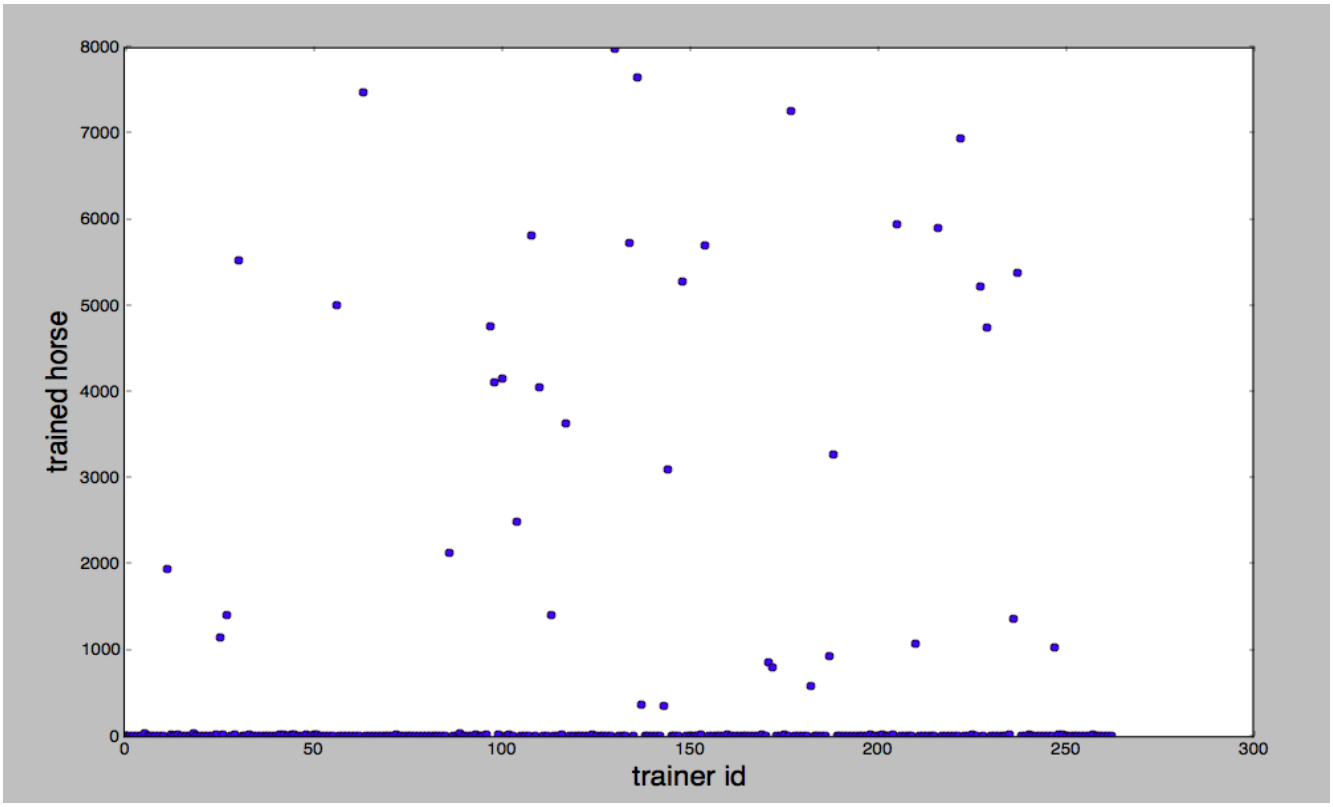

Figure 5. Trainer Distribution

JOCKEY NAME – Jockey name states the name of the person riding the horse in the race. There are 121 unique jockeys in the dataset. Jockey name column contains null and unmeaningful values which need to be imputed or modified before model training. Jockey name has many intangible measures associated to it like riding style, whip length, whip allotment and few other parameters. These measures aren't considered in our analysis but get second thought in real time betting. Jockey name is repetitive in dataset but not in one race. Jockey name column contains categorical data and so needs label encoding. This variable has significance in our analysis and has influence in real time betting. Following is the distribution of the number of horses that a trainer has trained.

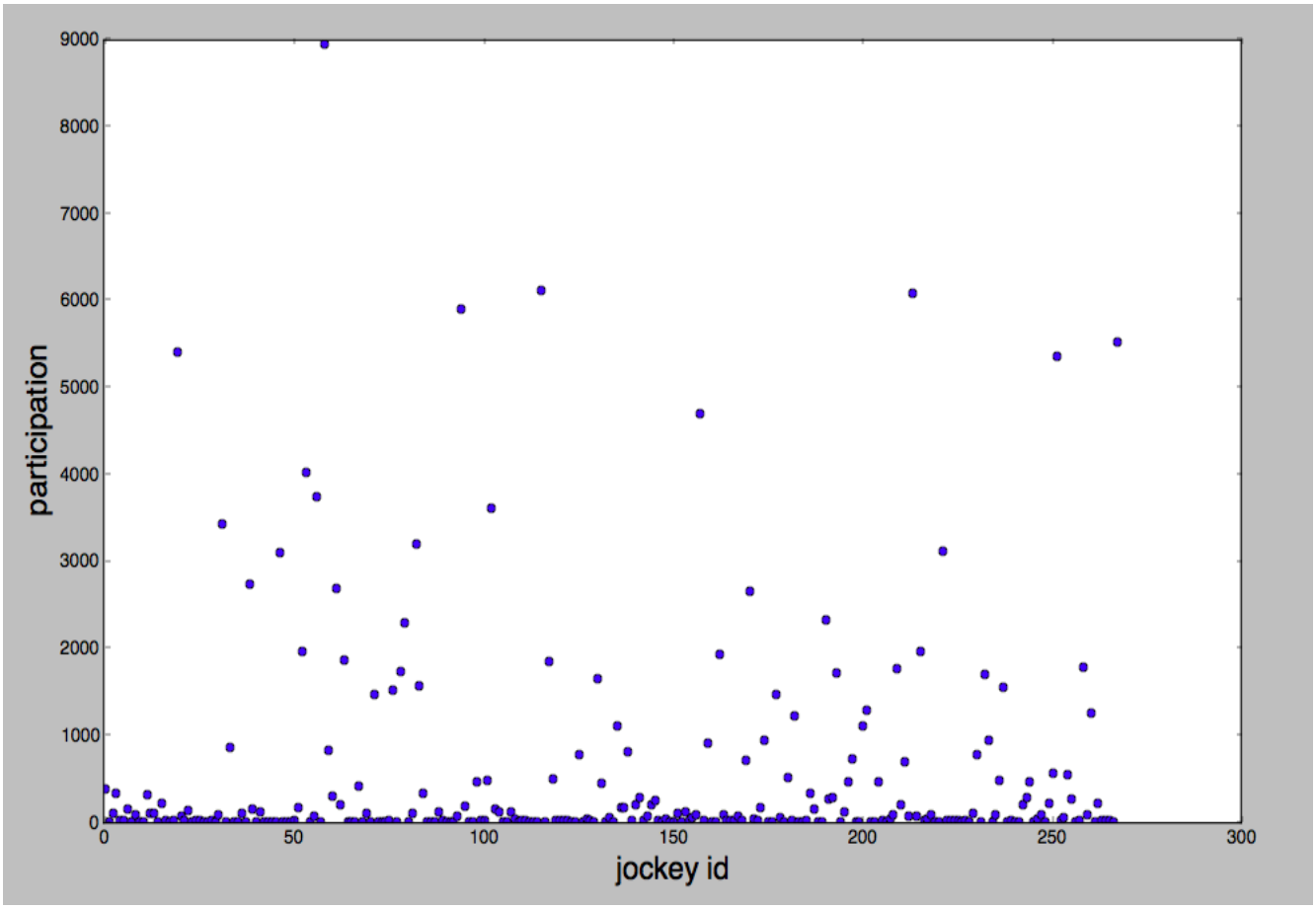

Figure 6. Jockey Distribution

Wt – Wt is the column containing continuous data. The values relate to jockey's real-time weight during the race. Mean value of the column is 55.61 units and std is 2.93. The min wt is 48.5 units and max wt is 66 units in the dataset. Luckily, to our help the wt column does not have missing values and so does not need preprocessing.

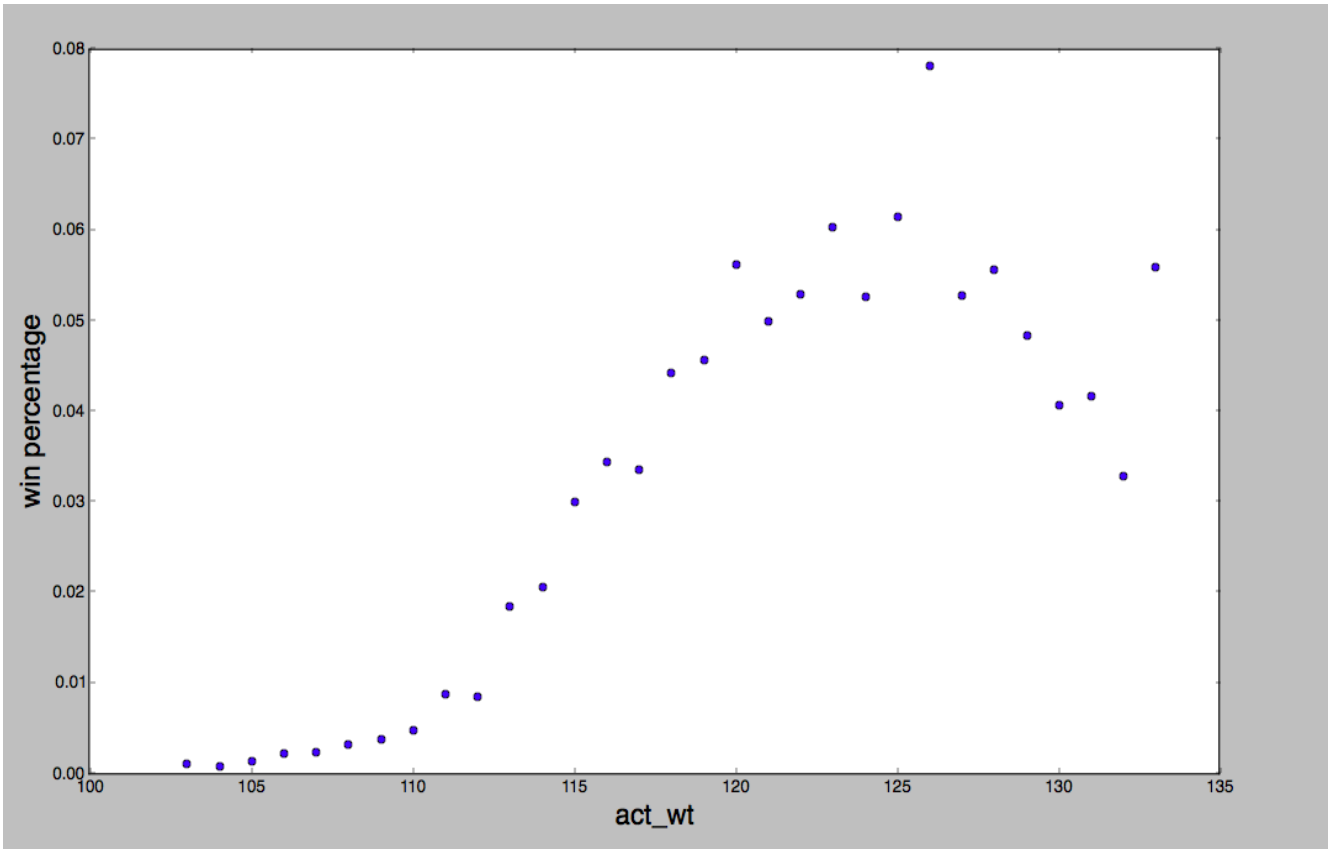

Figure 7. Weight Distribution

AL – This column includes floating values and missing values. It denotes the weight lost by the jockey from their last run. This column does not prove to be of much importance for the analysis and needs much preprocessing.

DR – Contains numeric values in discrete fashion. Value in this column denotes in which door the horse will be stalled for running. Initial analysis shows that Dr is significant in determining the results of the race and so is considered in the predictive analysis. The max number of horses ran in a race is 22. Luckily, to our help the Dr column does not have missing values and so does not need preprocessing. Following is the distribution of winning percentage of draws over the past 3 years.

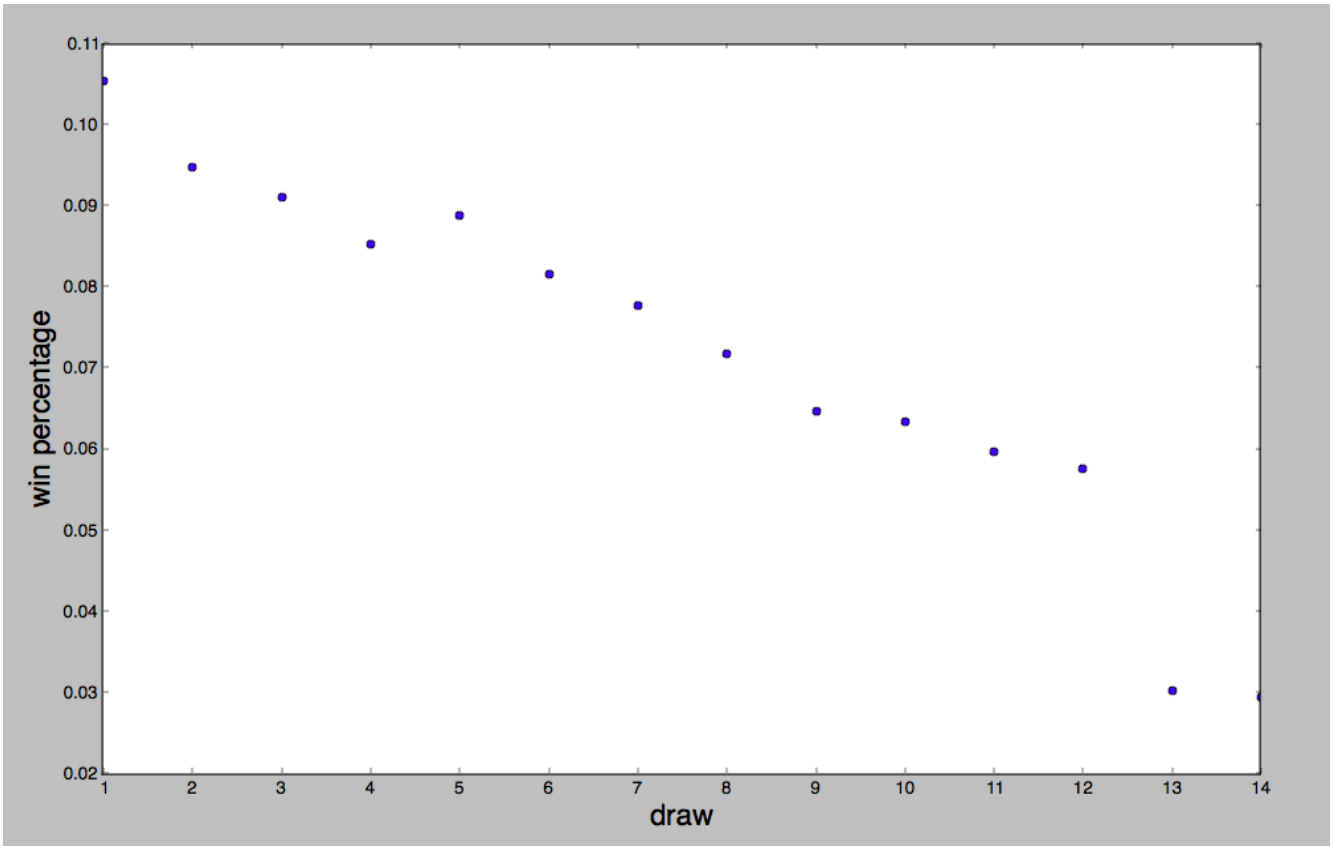

Figure 8. Door Draw Distribution

Sh – This is categorical column containing information about the shoe worn by the horse. S stands for Steel, A stands for Aluminum and U stands for Barefoot. 8578 horses have worn Aluminum horseshoe, 412 horses wore Steel horseshoe and 12 ran barefoot. Also, there are 30 missing values which were imputed in data cleaning.

```
In [12]: df.groupby('Sh').size()

Out[12]: Sh
         -       30
         A     8578
         S      412
         U       12
         dtype: int64
```

Won By – Won By column contains categorical and float(numerical) data and some missing values. The first value in race results is always null. The next value is the distance between winning horse and runner up which is measured in terms of horse length and then horse length between winner and second runner-up and so on. Some of the values in this column are NECK, ½, 3 1/2. Since, assumptions are also in place for this analysis which makes this column insignificant for our analysis.

Distance Win – Contains numeric continuous and missing values. The first value in race results is always null. The next value is the distance between winning horse and runner up which is measured in terms of horse length and so on. Some of the values in this column are 2,3.5,11.5. Since, assumptions are also in place for this analysis which makes this column insignificant for our analysis.

RATINGS – Ratings column contains continuous numeric values. Values denote the ratings for the horse in the race card. Data type of the column is int64. Ratings for every horse in the race card are different. Ratings for same horse in different race cards also may change depending on its form. Column contains missing values as well which are found with linear regression and imputed in the dataset.

```
In [15]: df.groupby('Ratings').size()
```

```
Out[15]: Ratings
-       1200
0         54
1          8
10        25
100       29
101        7
102        9
103        6
104        8
105       11
106       13
107       18
108        9
109       16
11        37
110       15
111       16
112        8
113       11
114       10
115       10
116        6
117        3
118        7
119        4
12        32
120        7
121        2
122        3
123        1
72        22
73        32
74        29
75        26
76        35
77        17
78        17
79        21
8         18
80        32
81        11
82        24
83        11
84        22
85        11
86        24
87        18
88        16
89         7
9         32
90        22
91        13
92        15
93        11
94        14
95        13
96        27
97         5
98        13
99        11
Length: 129, dtype: int64
```

Odds – Data type for Odds is object. Odds represent the chances of that horse winning the race. Odds have intangible elements like fear, greed and maybe some insider knowledge of how the horse may perform on that day associated with it. 'Favorite' term is used to point to the horse who has lowest odds and has baking from punters. Odds is a numeric continuous variable which needs transformation to floating values. This is a dynamic column which keeps changing based on punters backing but we would be only considering the final odds updated just before the race starts. Some of the values in this column are 25/1, 72/100, 13/1. If a punters bets $100 on a horse with Odds with 25/1 and if that horse wins then he gets back $2600 in return. This is the most significant variable in our analysis.

```
def convert_to_float(frac_str):
    try:
        return float(frac_str)
    except ValueError as error:
        num, denom = frac_str.split('/')
        try:
            leading, num = num.split(' ')
            whole = float(leading)
        except ValueError:
            whole = 0
        frac = float(num) / float(denom)
        return whole - frac if whole < 0 else whole + frac
```

Following is the distribution of race time odds:

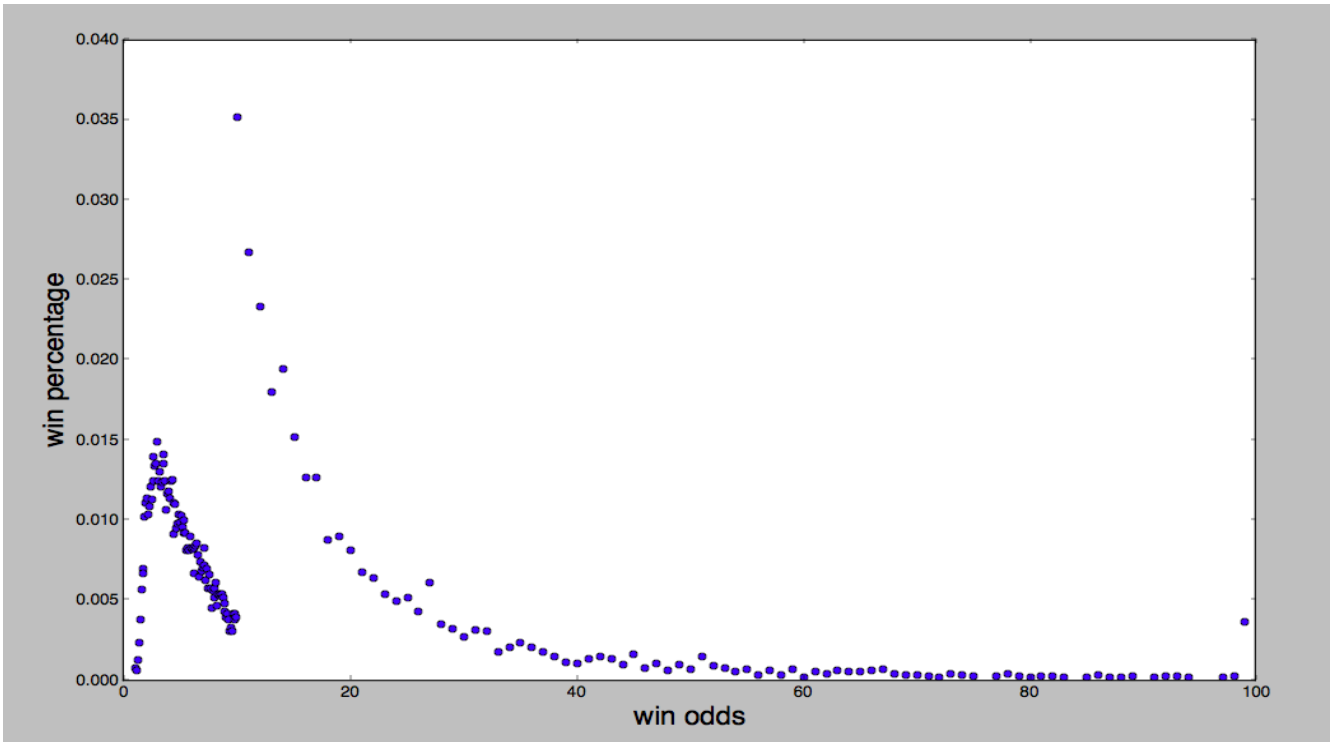

Figure 9. Odds Distribution

TIME – Column contains time recorded by the horse to complete the race. Time variates depending on the distance to cover. Races are of different lengths like 1000m, 1200m, 1800m, 2400m and 3200m. Length of the race track wasn't available for extraction and so time variable needs to be cut off from our analysis.

7.Data Preprocessing

Data Preprocessing was the major part in this project. Since, the web scrapped data had many missing values. When I looked at the data, the challenge seemed to be at its peak. Nearly 6% of the race time odds were missing and 14% of the ratings values were missing. Numerous discussions and readings were done to check what would happen if a bias is introduced or records or dropped. Race – time odds proved vital to developing a model with predictive power and range of odds between and other horses was sometimes very large. I would have had to proceed with large caveat in the dataset but after getting some guidance I decided to predict the missing values with linear regression and then imputed them in the dataset.

In [16]:
```
xm=df.iloc[:,[1,2,3,4,5,6]]
ym=df.iloc[:,7]
```

In [17]:
```
xt=nadf.iloc[:,[1,2,3,4,5,6]]
yt=nadf.iloc[:,7]
```

In [18]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(xm,ym,test_size=0.1,random_state =1)
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ver
sion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that t
he interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [19]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [20]:
```
regressor.fit(x_train,y_train)
```

Out[20]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [21]:
```
y_pred= regressor.predict(x_test)
```

In [ ]:
```
xt
```

In [22]:
```
y_prednadf= regressor.predict(xt)
```

In [23]:
```
y_prednadf
```

Out[23]:
```
array([35.66466678, 42.65131078, 42.39546776, ..., 32.7085033 ,
       41.57177789, 39.22035683])
```

In [24]:
```
len(y_prednadf)
```

Out[24]:
```
1518
```

In [25]:
```
nadfvalues = pd.DataFrame(y_prednadf)
```

In [26]:
```
nadfvalues.to_csv("ratingsvalues.csv")
```

8.Data Cleaning:

The data was structured but was so unclean that it needed much cleaning. It had unnecessary spacing and many non-null values which were imputed automatically by the spreadsheet. Such anamolies needed detection and termination. Following images explain the step by step cleaning process.

Figures below describes the dataset.

```
In [9]: df.describe(include='all')
```

Out[9]:

	Pl No	H.No	Horse	Desc	Trainer	Jockey	Wt	Al	Dr	Sh	Won By	Dist Win	Ratings	Odds	Time
count	9032	9032.000000	9032	9032	9029	9032	9032.000000	9032	9032.000000	9032	8040	7841.000000	9032	8645	8630
unique	24	NaN	1726	279	211	201	NaN	8	NaN	9	62	NaN	131	215	1908
top	1	NaN	INCENTIO	4y b g	Narendra Lagad	P Trevor	NaN	-	NaN	A	1/1	NaN	-	20/1	01:00.0
freq	956	NaN	28	1072	870	577	NaN	6647	NaN	8421	2027	NaN	1173	1106	75
mean	NaN	5.838795	NaN	NaN	NaN	NaN	55.614094	NaN	5.801484	NaN	NaN	9.220284	NaN	NaN	NaN
std	NaN	3.624271	NaN	NaN	NaN	NaN	2.939168	NaN	3.619835	NaN	NaN	7.607611	NaN	NaN	NaN
min	NaN	1.000000	NaN	NaN	NaN	NaN	48.500000	NaN	0.000000	NaN	NaN	0.500000	NaN	NaN	NaN
25%	NaN	3.000000	NaN	NaN	NaN	NaN	53.500000	NaN	3.000000	NaN	NaN	4.000000	NaN	NaN	NaN
50%	NaN	5.000000	NaN	NaN	NaN	NaN	55.500000	NaN	5.000000	NaN	NaN	7.250000	NaN	NaN	NaN
75%	NaN	8.000000	NaN	NaN	NaN	NaN	58.000000	NaN	8.000000	NaN	NaN	12.250000	NaN	NaN	NaN
max	NaN	22.000000	NaN	NaN	NaN	NaN	66.000000	NaN	22.000000	NaN	NaN	64.250000	NaN	NaN	NaN

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9032 entries, 0 to 9031
Data columns (total 9 columns):
Pl No       9032 non-null object
Horse       9032 non-null object
Trainer     9029 non-null object
Jockey      9032 non-null object
Wt          9032 non-null float64
Dr          9032 non-null int64
Sh          9032 non-null object
Ratings     9032 non-null object
Odds        8645 non-null object
dtypes: float64(1), int64(1), object(7)
memory usage: 635.1+ KB
```

Even though the dataset gets described with non -null objects but it is not we desire to have. Example, Pl No variable is supposed to contain only numeric values but since it is described as object means it includes textual data as well. Thus, this column needed cleaning. Similarly, variables Ratings, Sh and Odds demanded cleaning too.

Following figures explain the cleaning process for those variables.

```
df['Pl No']=df['Pl No'].str.strip()
df['Horse']=df['Horse'].str.strip()
df['Trainer']=df['Trainer'].str.strip()
df['Jockey']=df['Jockey'].str.strip()
df['Sh']=df['Sh'].str.strip()
df['Ratings']=df['Ratings'].str.strip()
df['Odds']=df['Odds'].str.strip()
```

```
In [8]: df['Pl No'].describe()

Out[8]: count     9032
        unique      22
        top          1
        freq       956
        Name: Pl No, dtype: object
```

```
In [9]: df.groupby('Pl No').size()

Out[9]: Pl No
        1      956
        10     442
        11     330
        12     233
        13     161
        14     105
        15      56
        16      32
        17      22
        18       9
        19       3
        2      955
        20       2
        21       1
        3      948
        4      927
        5      890
        6      811
        7      754
        8      661
        9      557
        W      177
        dtype: int64
```

```
df['Pl No'] = [1 if b=='1' else 0 for b in df['Pl No']]
```

In [ ]:
```
#df['Pl No']=df['Pl No'].replace('10',0,regex=True)
#df['Pl No']=df['Pl No'].replace('11',0,regex=True)
#df['Pl No']=df['Pl No'].replace('12',0,regex=True)
#df['Pl No']=df['Pl No'].replace('13',0,regex=True)
#df['Pl No']=df['Pl No'].replace('14',0,regex=True)
#df['Pl No']=df['Pl No'].replace('15',0,regex=True)
#df['Pl No']=df['Pl No'].replace('16',0,regex=True)
#df['Pl No']=df['Pl No'].replace('17',0,regex=True)
#df['Pl No']=df['Pl No'].replace('18',0,regex=True)
#df['Pl No']=df['Pl No'].replace('19',0,regex=True)
#df['Pl No']=df['Pl No'].replace('2',0,regex=True)
#df['Pl No']=df['Pl No'].replace('20',0,regex=True)
#df['Pl No']=df['Pl No'].replace('3',0,regex=True)
#df['Pl No']=df['Pl No'].replace('4',0,regex=True)
#df['Pl No']=df['Pl No'].replace('5',0,regex=True)
#df['Pl No']=df['Pl No'].replace('6',0,regex=True)
#df['Pl No']=df['Pl No'].replace('7',0,regex=True)
#df['Pl No']=df['Pl No'].replace('8',0,regex=True)
#df['Pl No']=df['Pl No'].replace('9',0,regex=True)
#df['Pl No']=df['Pl No'].replace('W',0,regex=True)
```

In [11]:
```
df.groupby('Pl No').size()
```

Out[11]:
```
Pl No
0    8076
1     956
dtype: int64
```

In [12]:
```
df.groupby('Sh').size()
```

Out[12]:
```
Sh
-      30
A    8578
S     412
U      12
dtype: int64
```

In [13]:
```
df['Sh']=df['Sh'].replace('-',np.nan,regex=True)
```

In [16]:
```
df['Ratings']=df['Ratings'].replace('-',np.nan,regex=True)
```

In [17]:
```
df.groupby('Ratings').size()
```

Out[17]:
```
Ratings
0      54
1       8
10     25
100    29
101     7
102     9
103     6
104     8
105    11
106    13
107    18
108     9
109    16
11     37
110    15
111    16
112     8
113    11
114    10
```

In [18]:
```
df["Ratings"].str.contains('-').any()
```

Out[18]:
```
False
```

In [19]:
```
df['Ratings'].isnull().any()
```

Out[19]:
```
True
```

9.Feature Selection

Assumptions are in place and so there are some columns or variables which do not stand any significance. First assumption, distance of the race is constant which made Won By, Distance Win and Time variables void. Secondly, Age is a complicated variable including age and gender and horse category, it was difficult to segregate that information and thus Desc had to be discarded from our analysis. Similarly, Allowance variable had many missing values and did not have signifcant correlation with the dependent variable so it was a better decision to exclude this variable as well. Likewise, Horse No was an extra column in the dataset which was insignificant and hold no importance in the analysis.

```
In [4]: list(df)
```

```
Out[4]: ['Pl No',
         'H.No',
         'Horse',
         'Desc',
         'Trainer',
         'Jockey',
         'Wt',
         'Al',
         'Dr',
         'Sh',
         'Won By',
         'Dist Win',
         'Ratings',
         'Odds',
         'Time']
```

```
In [5]: df = df.drop(['Al','H.No','Desc','Won By','Dist Win','Time'], axis=1)
        list(df)
```

```
Out[5]: ['Pl No', 'Horse', 'Trainer', 'Jockey', 'Wt', 'Dr', 'Sh', 'Ratings', 'Odds']
```

10.Variable transformation

The Machince Learning model only understands number which is why all the categorical variable need transformation to numeric. Columns Horse, Jockey, Trainer and Sh all need label encoding to be converted to numbers and be utilized in machine learning models. Following code makes the label encoding work.

```
df['Horse'] = labelencoderx.fit_transform(df.values[:, 1])
df['Jockey'] = labelencoderx.fit_transform(df.values[:, 3])
df['Trainer'] = labelencoderx.fit_transform(df['Trainer'].astype(str))
df['Sh'] = labelencoderx.fit_transform(df['Sh'].astype(str))
```

Odds are in the form of fractions and mixed fractions. Although these are numbers but need transformation to float values for the algorithm to understand. Following function makes the transformation.

```
def convert_to_float(frac_str):
    try:
        return float(frac_str)
    except ValueError as error:
        num, denom = frac_str.split('/')
        try:
            leading, num = num.split(' ')
            whole = float(leading)
        except ValueError:
            whole = 0
        frac = float(num) / float(denom)
        return whole - frac if whole < 0 else whole + frac
```

Figure 10. Odds conversion to float values

11. Approaches to train the model

Binary Approach – Pl No our dependent variable has values ranging from 1 to total number of horses in the race card. Binary Approach means to train the model in such a way it can predict whether a horse will win a race or no. For that, all the other classes were replaced by 0 except 1. This simply means, the horse that won the race will be have 1 against its row and all others would have 0.

```
df['Pl No'] = [1 if b=='1' else 0 for b in df['Pl No']]
```

```
In [11]: df.groupby('Pl No').size()

Out[11]: Pl No
         0    8076
         1     956
         dtype: int64
```

Multiclass Approach

Pl No dependent variable has values ranging from 1 to total number of horses in the race card. Multiclass classification means trying to predict the whole results. So, multi class approach emphasizes on getting all the horses classified into individual classes. If there are 10 horse in the race card then best horse gets predicted in class 1, second best horse in class 2,third best in class 3 and so on.

Finishing Time

We can train a regression model to estimate finishing time of a horse in a race card, and then the horse with lowest finish time could be predicted as winner.

Considering the time and resources available, I decided to move ahead with the binary approach which gives the probability of a horse winning the race and accordingly the punters can bet.

12.Model Training

There are number of machine learning algorithms in place for predictive analysis of which I have chosen one each from the Linear model, Ensemble model and Neural Network models. Logistic Regression is the linear model, Random Forest Classification is the ensemble method and Artificial Neural Network from the neural network family.

Linear Model

Logistic Regression is considered a generalized linear model because the outcome always depends on the sum of inputs and parameters. Logistic regression is an algorithm that learns a model for binary classification. Following is the logistic regression graph with sigmoid function.[8]

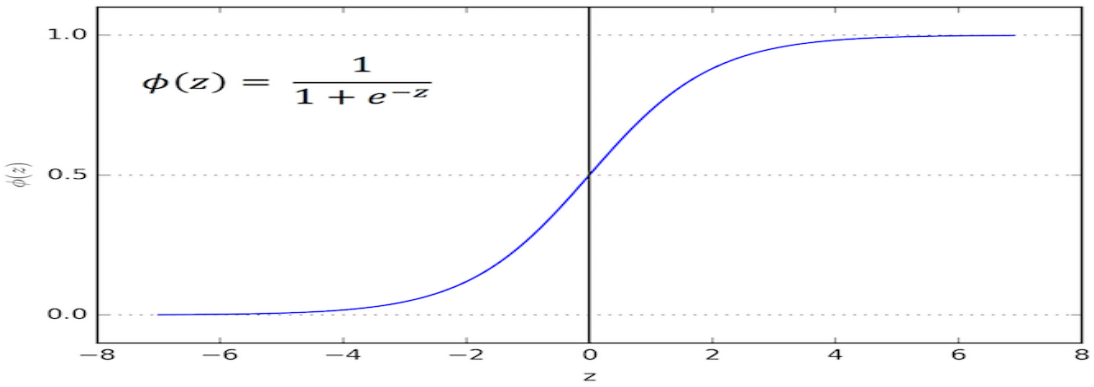

Figure 11. Logistic Regression with sigmoid function

One may begin to understand logistic regression by first considering a logistic model with given parameters, then seeing how the coefficients can be estimated ("regressed") from data. Consider a model with two predictors, x_1 and x_2; these may be continuous variables (taking a real number as value), or indicator functions for binary variables (taking value 0 or 1). Then the general form of the log-odds (here denoted by l) is:

$$l = \beta_0 + \beta_1 x_1 + \beta_2 x_2,$$

where the coefficients β_i are the parameters of the model. Note that this is a linear model: the log-odds l are a linear combination of the predictors x_1 and x_2, including a constant term β_0. The corresponding odds are the exponent:

$$o = b^{\beta_0+\beta_1 x_1+\beta_2 x_2}$$

where b is the base of the logarithm and exponent. This is now a non-linear model, since the odds are not a linear combination of the predictors.

Odds of $o = o : 1$ for an event are the same as probability of $o/(o+1)$, since happening o of the time and not happening 1 time corresponds to happening o times out of a total of $o+1$ events, so the corresponding probability is:

$$p = \frac{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}}{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}+1} = \frac{1}{1+b^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}}$$

[8] https://www.quora.com/Why-is-logistic-regression-considered-a-linear-model

Importing Libraries

Numpy, pandas, sklearn are the important libraries needed to train model in logistic regression. Following figure shows library imported:

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

Loading Data

Preprocessed and cleaned data is already exported in a csv file and can be imported into pandas data frame for further analysis. Exporting cleaned data makes it more convenient for usage in future model training. The data frame now contains selected feature variables and one dependent variable.

df = pd.read_csv("C:/Users/tanma/Mumbai15-16-17CleanedData.csv")

The cleaned dataset is sliced in two parts. First part contains rows excluding the last race. Second part includes only the last race.

Now the first needs to be split into train and validation data. I have kept 20% as validation set i.e. 20% of 8632 rows.

After train_test_split, logistic regression is imported from sklearn.linear library. Logistic Regression is then fit to the train set and validation test.

```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(xm,ym,test_size=0.2,random_state =1)
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
```

```
clf=logreg.fit(x_train,y_train)
```

Result

After fitting the model, the probabilities of test set are measured and unfortunately the first horse does not get max probability under class 1. So, the logistic model is unsuccessful in predicting the winner for the test race card.

```
In [25]: yt

Out[25]: 8632    1
         8633    0
         8634    0
         8635    0
         8636    0
         8637    0
         8638    0
         8639    0
         8640    0
         8641    0
         8642    0
         8643    0
         8644    0
         Name: Pl No, dtype: int64
```

```
In [21]: y_predunseenprob=logreg.predict_proba(xt)
```

```
In [22]: y_predunseenprob

Out[22]: array([[6.65670843e-01, 3.34329157e-01],
                [9.78150492e-01, 2.18495085e-02],
                [9.79102461e-01, 2.08975389e-02],
                [7.95987357e-01, 2.04012643e-01],
                [9.93969142e-01, 6.03085788e-03],
                [9.99983228e-01, 1.67724357e-05],
                [9.66784046e-01, 3.32159535e-02],
                [9.99921816e-01, 7.81836889e-05],
                [9.99998817e-01, 1.18342184e-06],
                [9.96924855e-01, 3.07514524e-03],
                [9.99987627e-01, 1.23734001e-05],
                [9.99913392e-01, 8.66082217e-05],
                [9.99998429e-01, 1.57055712e-06]])
```

Upsample Minority Class

Up-sampling is the process of randomly duplicating observations from the minority class to reinforce its signal. There are several heuristics for doing so, but the most common way is to simply resample with replacement.

Importing Libraries

Numpy, pandas, sklearn are the important libraries needed to train model in logistic regression. Following figure shows library imported:

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

Loading Data

Preprocessed and cleaned data is already exported in a csv file and can be imported into pandas data frame for further analysis. Exporting cleaned data makes it more convenient for usage in future model training. The data frame now contains selected feature variables and one dependent variable.

df = pd.read_csv("C:/Users/tanma/Mumbai15-16-17CleanedData.csv")

The cleaned dataset is sliced in two parts. First part contains rows excluding the last race. Second part includes only the last race. Now the first needs to be split into train and validation data. I have kept 20% as validation set i.e. 20% of 8632 rows.

After train_test_split, logistic regression is imported from sklearn.linear library. Logistic Regression is then fit to the train set and validation test.

```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(xm,ym,test_size=0.2,random_state =1)
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

First, we'll import the resampling module from Scikit-Learn:

```
from sklearn.utils import resample
```

Next, we'll create a new DataFrame with an up-sampled minority class. Here are the steps:

1. First, we'll separate observations from each class into different DataFrames.
2. Next, we'll resample the minority class **with replacement**, setting the number of samples to match that of the majority class.
3. Finally, we'll combine the up-sampled minority class DataFrame with the original majority class DataFrame.

```
# Separate majority and minority classes
df_majority = df[df['Pl No']==0]
df_minority = df[df['Pl No']==1]


# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=7709,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled['Pl No'].value_counts()
# 1    576
# 0    576
# Name: balance, dtype: int64
```

```
1    7709
0    7709
Name: Pl No, dtype: int64
```

Let's train another model using Logistic Regression, this time on the balanced dataset:

```
# Separate input features (X) and target variable (y)
y = df_upsampled['Pl No']
X = df_upsampled.drop('Pl No', axis=1)
```

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()

# Train model
clf_1 = logreg.fit(X, y)
```

```
# Predict on training set
pred_y_1 = clf_1.predict(X)
```

```
# Is our model still predicting just one class?
print( np.unique( pred_y_1 ) )
```

```
[0 1]
```

Result

After fitting the model, the probabilities of test set are measured and fortunately the first horse gets max probability under class 1. So, the logistic model with up sampled minority class is successful in classifying the first horse in class 1 and predicting the winner for the test race card.

```
predunseen = clf_1.predict(xt)
```

```
predunseen
```

```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

```
predunseenprob= logreg.predict_proba(xt)
```

```
predunseenprob
```

```
array([[2.38589455e-01, 7.61410545e-01],
       [7.99068194e-01, 2.00931806e-01],
       [7.93100233e-01, 2.06899767e-01],
       [3.46696301e-01, 6.53303699e-01],
       [9.27035105e-01, 7.29648952e-02],
       [9.99409587e-01, 5.90412548e-04],
       [7.30925550e-01, 2.69074450e-01],
       [9.98106438e-01, 1.89356230e-03],
       [9.99944825e-01, 5.51754720e-05],
       [9.53652347e-01, 4.63476526e-02],
       [9.99569529e-01, 4.30470777e-04],
       [9.97808863e-01, 2.19113699e-03],
       [9.99924802e-01, 7.51984606e-05]])
```

Ensemble Models

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. Empirically, ensembles tend to yield better results when there is a significant diversity among the models. Many ensemble methods, therefore, seek to promote diversity among the models they combine. Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees). Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt to *dumb-down* the models in order to promote diversity.

Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.[9]

[9] https://en.wikipedia.org/wiki/Random_forest

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

The number of samples/trees, *B*, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees *B* can be found using cross-validation, or by observing the *out-of-bag error*: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

Importing Libraries

Numpy, pandas, sklearn are the important libraries needed to train model in logistic regression. Following figure shows library imported:

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

Loading Data

Preprocessed and cleaned data is already exported in a csv file and can be imported into pandas data frame for further analysis. Exporting cleaned data makes it more convenient for usage in future model training. The data frame now contains selected feature variables and one dependent variable.

df = pd.read_csv("C:/Users/tanma/Mumbai15-16-17CleanedData.csv")

The cleaned dataset is sliced in two parts. First part contains rows excluding the last race. Second part includes only the last race.

Now the first needs to be split into train and validation data. I have kept 20% as validation set i.e. 20% of 8632 rows.

After train_test_split, RandomForestClassifier method is imported from sklearn.ensemble library. Random Forest Classifier is then fit to the train set and validation test.

```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(xm,ym,test_size=0.2,random_state =1)
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 1000, random_state = 0)
classifier.fit(x_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
# Predicting the Unseen set results
y_predun = classifier.predict(xt)
```

```
y_predprobun=classifier.predict_proba(xt)
y_predprobun
```

```
array([[0.433, 0.567],
       [0.857, 0.143],
       [0.998, 0.002],
       [0.906, 0.094],
       [0.982, 0.018],
       [0.995, 0.005],
       [0.91 , 0.09 ],
       [0.995, 0.005],
       [0.994, 0.006],
       [0.976, 0.024],
       [0.948, 0.052],
       [0.999, 0.001],
       [0.933, 0.067]])
```

```
yt
```

```
8632    1
8633    0
8634    0
8635    0
8636    0
8637    0
8638    0
8639    0
8640    0
8641    0
8642    0
8643    0
8644    0
Name: Pl No, dtype: int64
```

Artificial Neural Network

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.[10]

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

[10] https://en.wikipedia.org/wiki/Artificial_neural_network

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

Importing Libraries

Numpy, pandas, sklearn are the important libraries needed to train model in logistic regression. Following figure shows library imported:

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

Loading Data

Preprocessed and cleaned data is already exported in a csv file and can be imported into pandas data frame for further analysis. Exporting cleaned data makes it more convenient for usage in future model training. The data frame now contains selected feature variables and one dependent variable.

df = pd.read_csv("C:/Users/tanma/Mumbai15-16-17CleanedData.csv")

The cleaned dataset is sliced in two parts. First part contains rows excluding the last race. Second part includes only the last race.

Now the first needs to be split into train and validation data. I have kept 20% as validation set i.e. 20% of 8632 rows.

```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(xm,ym,test_size=0.2,random_state =1)
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Now importing dependent libraries and building Neural network.

```
# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype
from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
# Initialising the ANN
classifier = Sequential()
```

```
# Adding the input layer and the first hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 8))
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `Dense` call to the Keras 2 API: `
Dense(activation="relu", input_dim=8, units=6, kernel_initializer="uniform")`
```

```
# Adding the second hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `Dense` call to the Keras 2 API: `
Dense(activation="relu", units=6, kernel_initializer="uniform")`
```

```
# Adding the output layer
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

```
C:\Users\tanma\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `Dense` call to the Keras 2 API: `
Dense(activation="sigmoid", units=1, kernel_initializer="uniform")`
```

```
# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# Fitting the ANN to the Training set
classifier.fit(x_train, y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
6905/6905 [==============================] - 1s 139us/step - loss: 0.3406 - acc: 0.8915
Epoch 2/100
6905/6905 [==============================] - 1s 78us/step - loss: 0.2807 - acc: 0.8915
Epoch 3/100
6905/6905 [==============================] - 1s 85us/step - loss: 0.2756 - acc: 0.8915
Epoch 4/100
6905/6905 [==============================] - 1s 78us/step - loss: 0.2728 - acc: 0.8918
Epoch 5/100
6905/6905 [==============================] - 1s 75us/step - loss: 0.2717 - acc: 0.8918
Epoch 6/100
6905/6905 [==============================] - 1s 75us/step - loss: 0.2720 - acc: 0.8921
Epoch 7/100
6905/6905 [==============================] - 1s 76us/step - loss: 0.2723 - acc: 0.8914
Epoch 8/100
6905/6905 [==============================] - 1s 73us/step - loss: 0.2737 - acc: 0.8917
Epoch 9/100
6905/6905 [==============================] - 1s 75us/step - loss: 0.2701 - acc: 0.8918
```

Results

```
# Predicting the Test set results
y_predunseen = classifier.predict(xt)
```

```
y_predunseen
```

```
array([[3.05431962e-01],
       [1.31673645e-02],
       [1.97729189e-02],
       [1.94574326e-01],
       [6.39773440e-03],
       [1.87148992e-03],
       [3.36853564e-02],
       [8.74240242e-04],
       [6.50754955e-05],
       [3.09634930e-03],
       [3.89554334e-05],
       [9.20101884e-04],
       [2.75957318e-05]], dtype=float32)
```

13.Future Scope

I plan to group the previous horse race results for a horse and append it to the same row thus increasing independent variables. This will provide more variables for the model to train and predict better results.

After this I plan to develop a front end which will allow the user to import real time race card and get predictions for the race.

14.Conclusion

Logistic Regression with Up sampled minority class and Random Forest Classifier prove vital in predicting the winner for the test race card and can be tested on more real time race data.

15.Acknowledgement

I would like to acknowledge Dr. Sergei Schreider, Assistant Professor of Professional Practice at Rutgers Business School for weekly reviewing the project progress and suggesting machine learning models to best fit the analysis.