Design Document for Machine Learning for March Madness

Alex Hoffer, Chongxian Chen, and Jacob Smith
Team Name: Stat Champs

TABLE OF CONTENTS

I. Introduction

Stat Champs
December 2, 2016

AFTER making reasoned decisions about which technologies to use to implement this module, we must now consider how these technologies will work together. This design document intends to explain how the three technologies each of the Stat Champs have selected to use will complete their assigned task. It will do so through the use of diagrams followed by paragraphs describing what the technology must do and why it must do it. The first three designs were produced by Alex Hoffer. These first three designs are chiefly concerned with the usability of the service, specifically the design of the graphical user interface, the instructions that will be presented to the user, and the display of the machine learning bracket that is generated. In technologies 4 and 5 Jake Smith will talk about how we will obtain and store the sports statistics for easy use by our machine learning algorithms. It is our belief that in order for the tool to be effective in educating students it must be well designed and easy to understand. Chongxian Chen will be responsible for designing and implementing machine learning algorithm. Technologies 6, 7 and 8 will be discussing technology involved in designing machine learning algorithm, namely algorithm library, statistics model and cloud server for hosting the project

II. Glossary

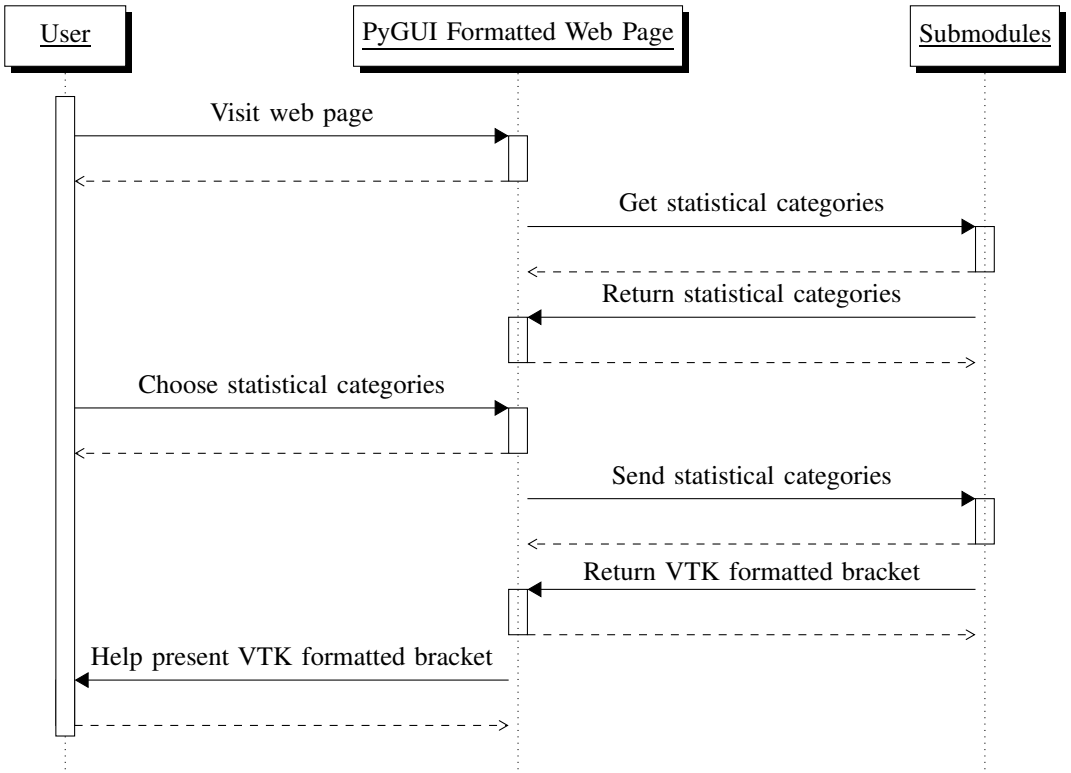
PyGUI: Graphical user interface API designed specifically for use with the Python programming language.
VTK: Data visualization API that will be used with the Python programming language.
Client-server architecture: Website format that consists of a user sending and receiving data to a server which also sends and receives data.
Webix: JavaScript API that is designed to support graphical user interfaces.
AWS: Amazon Web Service.
EC2: Amazon Elastic Compute Cloud.
SSH: Secure Shell.
RDS: Amazon Relational Database Service.
2P = 2-Point Field Goals
2P Percentage = 2-Point Field Goal Percentage
2PA = 2-Point Field Goal Attempts
3P = 3-Point Field Goals
3P Percentage = 3-Point Field Goal Percentage
3PA = 3-Point Field Goal Attempts
AST = Assists
AST Percentage = Assist percentage is an estimate of the percentage of teammate field goals a player assisted while he was on the floor.
Award Share = The formula is (award points) / (maximum number of award points).
BLK = Blocks
BLK Percentage = Block percentage is an estimate of the percentage of opponent two-point field goal attempts blocked by the player while he was on the floor.
BPM = Box Plus/Minus is a box score estimate of the points per 100 possessions that a player contributed above a league-average player, translated to an average team.
DRB = Defensive Rebounds
DRB Percentage = Defensive rebound percentage is an estimate of the percentage of available defensive rebounds a player grabbed while he was on the floor.
DRtg = Defensive Rating for players and teams it is points allowed per 100 possessions.
DWS = Defensive Win Shares
eFG Percentage = Effective Field Goal Percentage; the formula is (FG + 0.5 * 3P) / FGA. This statistic adjusts for the fact that a 3-point field goal is worth one more point than a 2-point field goal.
FG = Field Goals (includes both 2-point field goals and 3-point field goals)
FG Percentage = Field Goal Percentage; the formula is FG / FGA.
FGA = Field Goal Attempts (includes both 2-point field goal attempts and 3-point field goal attempts)
FT = Free Throws
FT Percentage = Free Throw Percentage; the formula is FT / FTA.
FTA = Free Throw Attempts
Four Factors = Dean Oliver's "Four Factors of Basketball Success
G = Games
GB = Games Behind
GmSc = Game Score: was created to give a rough measure of a player's productivity for a single game. The scale is similar

to that of points scored, (40 is an outstanding performance, 10 is an average performance, etc.)
GS = Games Started
MP = Minutes Played
MOV = Margin of Victory
ORtg = Offensive Rating for players it is points produced per 100 posessions, while for teams it is points scored per 100 possessions.
ORB = Offensive Rebounds
ORB Percentage = Offensive rebound percentage is an estimate of the percentage of available offensive rebounds a player grabbed while he was on the floor.
Pace = Pace factor is an estimate of the number of possessions per 48 minutes by a team. (Note: 40 minutes is used in the calculation for the WNBA.)
PER = Player Efficiency Rating The PER sums up all a player's positive accomplishments, subtracts the negative accomplishments, and returns a per-minute rating of a player's performance
PF = Personal Fouls
Poss = This formula estimates possessions based on both the team's statistics and their opponent's statistics, then averages them to provide a more stable estimate.
PProd = Points Produced
SOS = Strength of Schedule; a rating of strength of schedule. The rating is denominated in points above/below average, where zero is average
SRS = Simple Rating System; a rating that takes into account average point differential and strength of schedule.
STL = Steals
STL Percentage = Steal Percentage is an estimate of the percentage of opponent possessions that end with a steal by the player while he was on the floor.
Stops = Measure of individual defensive stops
TOV = Turnovers
TOV Percentage = Turnover percentage is an estimate of turnovers per 100 plays.
TRB = Total Rebounds
TRB Percentage = Total rebound percentage is an estimate of the percentage of available rebounds a player grabbed while he was on the floor.
TS Percentage = True shooting percentage is a measure of shooting efficiency that takes into account field goals, 3-point field goals, and free throws.
TSA = True Shooting Attempts
Usg Percentage = Usage percentage is an estimate of the percentage of team plays used by a player while he was on the floor.
Win Probability = The estimated probability that Team A will defeat Team B in a given matchup.

III. Designs

Design viewpoint 1: Using PyGUI for GUI of webpage

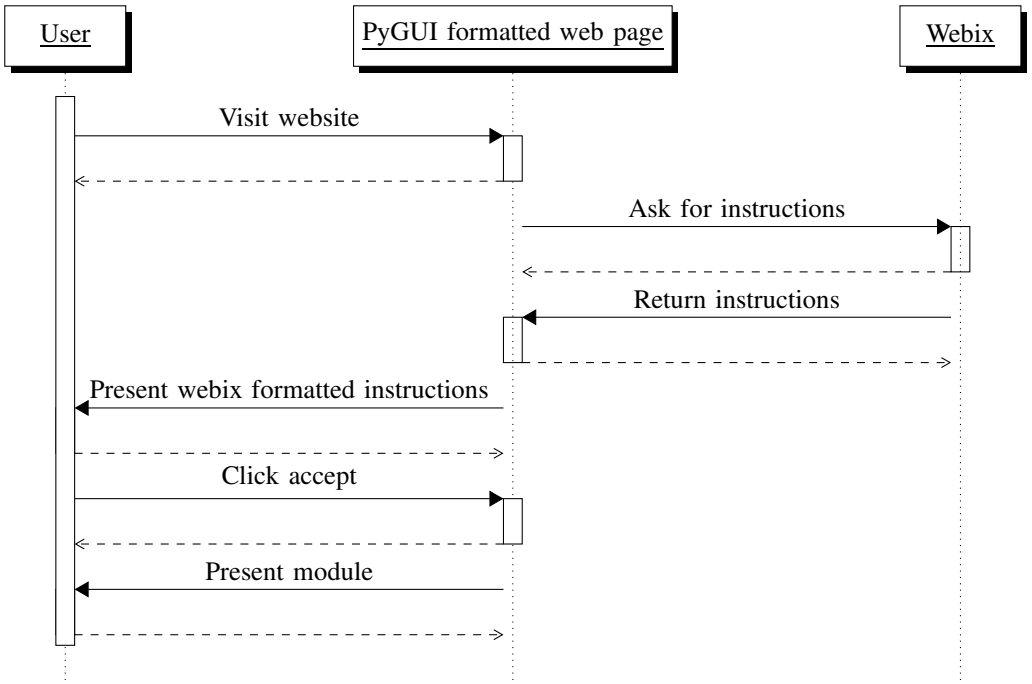
Design view 1:



Design Rationale: The module needs a web page to operate. This web page needs to be presented in a way that is pleasing to the eye and easy to use, because our intended users are biochemists, not computer scientists. This means the page needs to be as non-esoteric as possible. To achieve this, we will be using PyGUI to format our web page. The precondition of this sequence diagram is that the user has a browser. The postcondition of this diagram are that the user is presented with a bracket that resulted from collaboration between PyGUI and VTK. The flow of events from the perspective of PyGUI is quite simplistic. While a typical client-server architecture will allow us to transfer data back and forth between our submodules, PyGUI must make this data transmission appear as convenient as possible to the user.

Design viewpoint 2: Using Webix to present instructions for the module

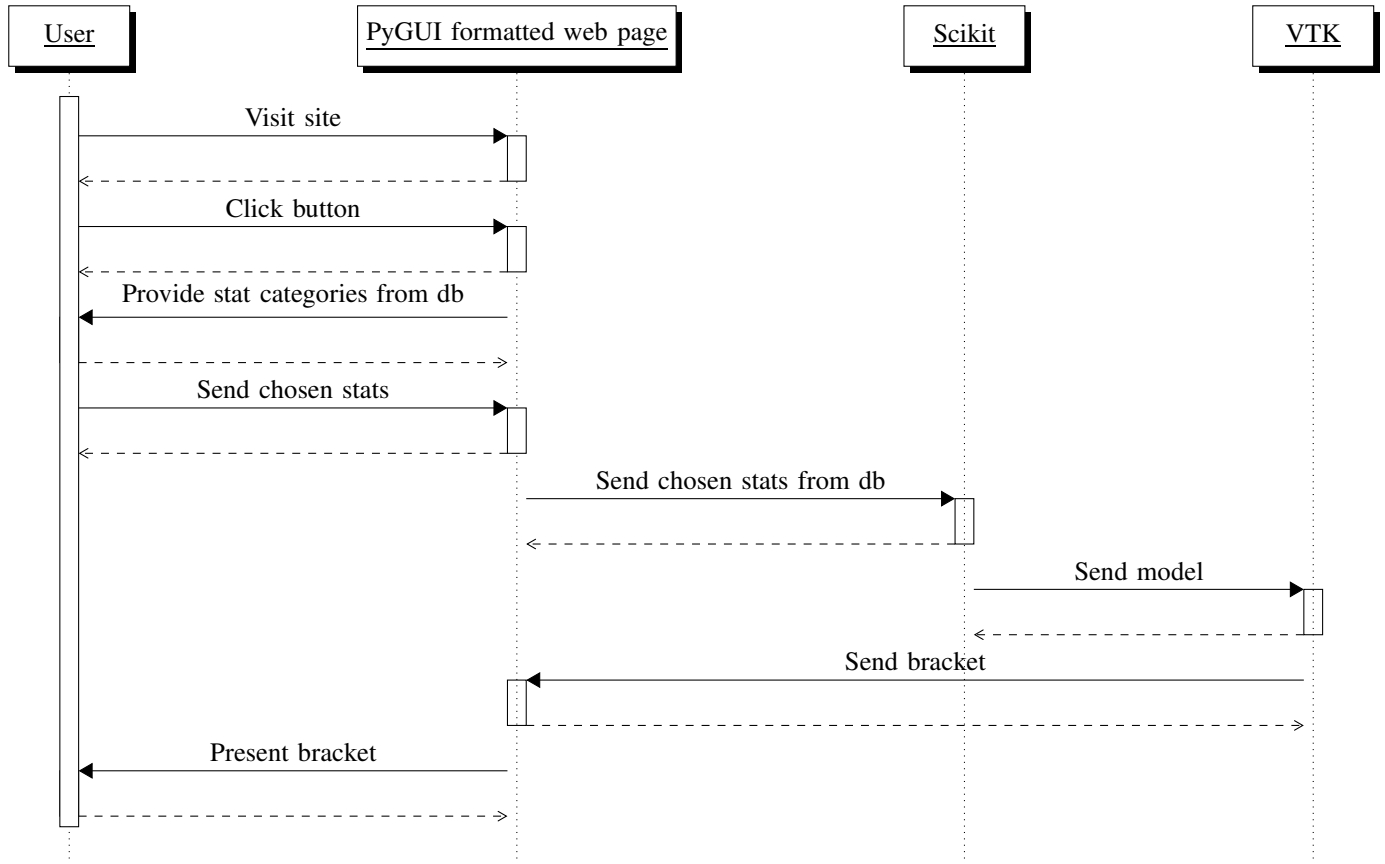
Design view 2:



Design Rationale: The module needs to present instructions to the user which informs them how to use the service. These instructions should appear seamlessly and should be visually inoffensive. Our main aim is reducing the chance of these instructions appearing unclear and thus maximizing utility. To achieve this, we will be using Webix, which will interact with our client/server code as well as with PyGUI. The precondition of this sequence diagram is that the user has a browser and has loaded the web page. The postconditions are that the user has seen the instructions (hopefully having read them) and has clicked accept to begin the service. The flow of events of the average use case are as follows: 1) the user visits the web page 2) the user is presented with instructions on properly using the module 3) the user reads the instructions 4) the user clicks accept 5) the module is presented. Note that the PyGUI formatted page in this diagram is a bit vague. PyGUI, of course, cannot handle the entire module. The web page will be calling a number of other technologies to present the module, but these technologies are not necessary when only considering how Webix will be used.

Design viewpoint 3: Generating bracket using VTK

Design view 3:



Design Rationale: The most essential function of the entire module is presenting a machine learned March Madness bracket. However, the process of machine learning is completely separate from the design of the bracket which will be displayed. Once Scikit generates a machine learned model that reflects the user's chosen statistics, it must communicate to VTK what this data is, and VTK will handle the bracket generation. In order for VTK to produce a bracket for display, a number of preconditions must occur. First, the user must have read and accepted the instructions. Then, the user must have selected data produced by a database. This data must be passed to Scikit, which generates a learned model. The postconditions of this sequence diagram is that the user will have seen the March Madness bracket that results from their chosen statistics. The flow of events for the average use case is as follows: 1) The user reads and accepts the instructions 2) The user selects statistics 3) The server sends these statistics to a machine learning submodule 4) The submodule generates a model 5) The submodule passes the model to VTK 6) VTK transforms the model into a bracket 7) A collaboration of VTK, PyGUI, and the server produce this bracket for the user to view.

Design viewpoint 4: Collection of data for the database

Design Rationale: The Machine Learning algorithms we want to work with get more accurate the more examples and stats you feed into it. Therefore, for the data collection we need to include as many possible data points as possible. We will do this by collecting stats from every major stat website and also pull more advanced data from the hoop-math which breaks down each players moves into more specialized data points. For example, hoop-math with break down a players shots from just saying he shot a 2 pointer to telling you where the shot was and how close a defender was to him at that moment. The design to grab the stats is to manually download all the csv files I can from Sports-Reference and upload them to the database then once I am caught up with the current season I will create a python script that takes the stats from all the daily games and uploads it into the database.

Design viewpoint 5: Database Design Design view 4: Couldnt get UML database design to post correctly. Design Rationale:

For designing the database I have decided to have multiple connected databases. The fist will be a running tally for all the players stats, everything from FG percentage to player efficiency and more. The second will be all the team game logs with their opponents stats for that game as well. That way we can do look ups for past matchups and analyze how both teams and players did against each other and apply that for future matchups. Then for the final database that will be connected to the game logs will be the advanced game player stats for example it will have play by play stats like who passed to who, when and who shot the ball, from where, and who was guarding him. The user will be able to choose between either all three databases in there bracket analysis or just two or even one. Each bracket should be able to bring in stats that should change

the outcome of the machine learning predictions.

Design viewpoint 6: SciKit Learn Machine Learning Model Design Rationale:

The project will be implementing the prediction model using python machine learning library SciKit-Learn. More specifically, we will be using the supervised learning regression model.

The model will be starting with simple and straightforward input first, and then gradually add more complex and hard-to-predict input. The benefit of starting with straightforward input is that we can easily verify the prediction accuracy of our model. For instance, the most straightforward input could be the match results between two teams in recent years. In most situation the team that wins significantly more in the recent matches should have a higher chance of winning current match. After confirming that our model could predict well on basic inputs, we can then add more complex inputs that may also affect the result of the game like home team status, weather and rest status.

After verifying that our model?s prediction accuracy is satisfying, we will be working on enable and disable categories of inputs to meet our project?s education purpose. When enabling all or most categories of inputs, we will be expecting our model to be more accurate. With trivial category of input, our model may show very inaccurate prediction that may even contradict to recent match results. The education purpose of our project will allow users to see a difference when they choose different category of data.

When the model is basically complete, we will be testing it with actual matches, particularly the march madness event. We will be continuing modifying our algorithm to enhance the prediction accuracy with live matches.

Timeline: Training model with basic inputs: Winter Week 1 to Week 3. Training model with more categories of inputs: Winter Week 4 to Week 6. Enable and disable category of inputs: Winter Week 7 to Week 9. Testing and improving the model with March Madness: Spring Week 2 to Spring Week 5.

Design viewpoint 7: Amazon Web Service(AWS) to host the database and computing power Design Rationale:

We will be hosting our database and computing machine on Amazon Web Service. AWS is free for the first 12 months of registration with some limitation of use. After researching on AWS website, we will be able to host one linux instance free with Amazon EC2. We will also be able to hosting a free MySQL database with limited I/O from Amazon RDS. Although the I/O times are limited, but it is large enough for our project(10,000,000 I/Os). Hosting our computing power and database on the cloud is a good idea because we will be able to easily resize our computing power and database with AWS infrastructure when our project grows to a bigger size.

To start with AWS free tier for 12 months , we will be creating a group account. First we will be going to EC2 and host a Linux instance. We will have to choose a region among Amazon?s global servers. The one that is most convenient to us will be US West(Oregon). The operating system we will be using will be Ubuntu Server. Ubuntu is very popular among developers and have a large supporting community if we have trouble. Python can run seamlessly on Ubuntu terminal. For the instance type, we have only one option. T2.micro with one core cpu, one Gigabyte Ram and 8 Gigabyte Storage. The Ram is not very big so we may need to be careful with our ram usage or upgrade it in the future.

After creating the EC2 instance, we will moving forward to create the RDS database instance. We will be using relational database MySQL to store our data. The free tier also provide us with one core cpu, one Gigabyte Ram and 8 Gigabyte Storage. That should be enough for the teams in NCAA. We can also easily upgrade it when our project grows. When completing the setup of the database on AWS, we need to manually connect it to a database management tools so we can create tables and manage data. A convenient cross-platform tool we can use will be MySQL workbench. It is open-sourced, free and works very well on most platforms including Windows, Mac and Linux. We will be using SSH to connect the database on AWS to MySQL workbench. After that we can manage it like we have learned in class. Running SQL queries or create table using MySQL workbench?s graphical interface.

Timeline: Creating EC2 and RDS instance on AWS: Winter week 1 to week 2. Connecting to EC2 and RDS instance: Winter Week 3 to Week 4.

Design viewpoint 8: Statistics Model Design Rationale:

With the Statistics Libraries in Python like Py-Statiscs and a large amount of input data in our library, our statistics model is expected to give a good estimate about the weight of each factor in our prediction model. This factor will vary for different teams, but we will have a general statistics equation. After providing it with a large amount of data for each teams, the equation is expected to give respective weight for each team. And then we can supply these weight to our machine learning model to get the final result.

First we will be implementing the equation with basic inputs matching results. With only matching results, the equation may doesn?t make too much sense to estimate the weight since there is nothing to compare with. Then we should add a trivial data to the equation so we can tell the difference. We will be expecting the trivial data to have a significantly less weight than recent match results. For the trivial data, the candidates are weather, team colors, etc. We will try the team colors first since that is an easy to collect data category for the beginning of our project.

After the basic statistics model makes sense with the data category we provide, we will be adding more category of data like player info, home game status, coach info etc. With a large amount of data, we should be able to a weight of each of these categories. Then we can apply these weights in our prediction model. We will be expecting to see our model become more

accurate. With more important categories of data added, our prediction should be more confident and we will be testing it with the march madness results next Spring. With the new data, we will be testing the accuracy of our predictions and improving it.

Timeline: Implement statistics model with basic inputs: Winter Week 1 to Week 3. Implement statistics model with more categories of inputs: Winter Week 4 to Week 6. Enable and disable category of inputs and generate perspective weight: Winter Week 7 to Week 9. Testing and improving the statistics model with March Madness: Spring Week 2 to Spring Week 5.

IV. AGREEMENT

Client

Developer

Developer

Developer

REFERENCES

[1] Kivy: Cross-platform Python Framework for NUI, *Kivy*. [Online]. Available: https://kivy.org/. [Accessed: 14-Nov-2016].

[2] D. Bolton, 5 Top Python GUI Frameworks for 2015 - Dice Insights, *Dice*, Feb-2016. [Online]. Available: http://insights.dice.com/2014/11/26/5-top-python-guis-for-2015/. [Accessed: 14-Nov-2016].

[3] JavaScript Framework and HTML5 UI Library for Web App Development-Webix, *Webix*. [Online]. Available: https://webix.com/. [Accessed: 14-Nov-2016].

[4] UKI, *Best Web Frameworks*. [Online]. Available: http://www.bestwebframeworks.com/web-framework-review/javascript/117/uki/. [Accessed: 14-Nov-2016].

[5] J. Shore, An Unconventional Review of AngularJS, *Let's Code Javascript*, 14-Jan-2015. [Online]. Available: http://www.letscodejavascript.com/v3/blog/2015/01/angular_review. [Accessed: 14-Nov-2016].

[6] F. Lungh, Notes on Tkinter Performance, *Effbot*, 14-Jul-2002. [Online]. Available: http://effbot.org/zone/tkinter-performance.htm. [Accessed: 14-Nov-2016].

[7] VTK-Enabled Applications, *VTK*. [Online]. Available: http://www.vtk.org/. [Accessed: 14-Nov-2016].

[8] Wax GUI Toolkit, *Python*. [Online]. Available: https://wiki.python.org/moin/wax. [Accessed: 14-Nov-2016].

[9] Amazon Machine Learning, *Amazon*. [Online]. Available: https://aws.amazon.com/machine-learning/. [Accessed: 14-Nov-2016].

[10] "SciKit Learn,Machine Learning in Python, *scikit-learn*. [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 14-Nov-2016].

[11] "Pylearn2 devdocumentation, *Pylearn2*. [Online]. Available: http://deeplearning.net/software/pylearn2/. [Accessed: 14-Nov-2016].

[12] "ONID - OSU Network ID, *Oregon State University*. [Online]. Available: http://onid.oregonstate.edu. [Accessed: 14-Nov-2016].

[13] "Python statistics Mathematical statistics functions, *Python*. [Online]. Available: https://docs.python.org/3/library/statistics.html. [Accessed: 14-Nov-2016].

[14] "Statistical functions (scipy.stats), *scipy.stats*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/stats.html. [Accessed: 14-Nov-2016].

[15] "Python Data Analysis Library, *Pandas*. [Online]. Available: http://pandas.pydata.org. [Accessed: 14-Nov-2016].