Soccer players, BMI, and birthday

Acquire data

Data was acquired from SOFIFA.com via the Python scraping program “soccer players.py”.

Extract only relevant data

```
In[151]:= SetDirectory[NotebookDirectory[]];

In[152]:= data = Import["soccer data.csv"];

In[ ]:= Position[data[[1]], "Birthdate"]

Out[ ]= {{27}}

In[ ]:= birthdates =
  ToExpression[Table[StringSplit[StringReplace[data[[i, 27]], {"(" → "", ")" → "",
       "," → "", "Jan" → "1", "Feb" → "2", "Mar" → "3", "Apr" → "4", "May" → "5",
       "Jun" → "6", "Jul" → "7", "Aug" → "8", "Sep" → "9", "Oct" → "10",
       "Nov" → "11", "Dec" → "12"}], "/"], {i, 2, Length[data]}]];

In[ ]:= InputForm[birthdates[[1 ;; 10]]]

Out[ ]//InputForm=
{{6, 24, 1987}, {2, 5, 1985}, {2, 5, 1992}, {11, 7, 1990}, {6, 28, 1991}, {1, 7, 1991},
 {9, 9, 1985}, {1, 24, 1987}, {7, 28, 1993}, {1, 7, 1993}}

In[153]:= heights = ToExpression[StringSplit[data[[2 ;; All, 28]], "'"]];

In[154]:= weights = ToExpression[StringReplace[data[[2 ;; All, 29]], "lbs" → ""]];

In[ ]:= Length[heights]

Out[ ]= 17630

In[ ]:= InputForm[heights[[-1]]]

Out[ ]//InputForm=
{5, 9}

In[ ]:= InputForm[weights[[-1]]]

Out[ ]//InputForm=
165

(*lambdaPis= λ Piscium STAR ;*)

In[ ]:= (*firstaries=UnitConvert[
  StarData[lambdaPis,EntityProperty["Star","RightAscension",{"Date"→Now}]],"Degrees"]*)
```

In[•]:=
```
birthdates[[1, {3, 1, 2}]]
```

Out[•]=
```
{1987, 6, 24}
```

Calculate Tropical sun degrees and signs

```
sundegrees =
  Round[QuantityMagnitude[Table[(*Mod[*)UnitConvert[SunPosition[ London CITY , Join[
        birthdates[[i, {3, 1, 2}]], {12, 0, 0}], CelestialSystem -> "Equatorial"][[1]],
      "Degrees"](*,Quantity[360,"Degrees"]]*), {i, 1, Length[birthdates]}]]];
```

In[155]:=
```
sundegrees = Import["sundegrees.m"];
```

```
(*cusplist={};
For[i=1,i≤Length[sundegrees],i++,
 If[sundegrees[[i]]==0||sundegrees[[i]]==30||
   sundegrees[[i]]==60||sundegrees[[i]]==90||sundegrees[[i]]==120||
   sundegrees[[i]]==150||sundegrees[[i]]==180||sundegrees[[i]]==210||
   sundegrees[[i]]==240||sundegrees[[i]]==270||sundegrees[[i]]==300||
   sundegrees[[i]]==330||sundegrees[[i]]==360,AppendTo[cusplist,i]]];*)

(*cusplist*)
```

Out[•]=
```
{13, 38, 47}
```

In[156]:=
```
sunsigns = ConstantArray[0, Length[sundegrees]];
For[i = 1, i ≤ Length[sundegrees], i++,
  If[sundegrees[[i]] > 0 && sundegrees[[i]] < 30, sunsigns[[i]] = 1];
  If[sundegrees[[i]] > 30 && sundegrees[[i]] < 60, sunsigns[[i]] = 2];
  If[sundegrees[[i]] > 60 && sundegrees[[i]] < 90, sunsigns[[i]] = 3];
  If[sundegrees[[i]] > 90 && sundegrees[[i]] < 120, sunsigns[[i]] = 4];
  If[sundegrees[[i]] > 120 && sundegrees[[i]] < 150, sunsigns[[i]] = 5];
  If[sundegrees[[i]] > 150 && sundegrees[[i]] < 180, sunsigns[[i]] = 6];
  If[sundegrees[[i]] > 180 && sundegrees[[i]] < 210, sunsigns[[i]] = 7];
  If[sundegrees[[i]] > 210 && sundegrees[[i]] < 240, sunsigns[[i]] = 8];
  If[sundegrees[[i]] > 240 && sundegrees[[i]] < 270, sunsigns[[i]] = 9];
  If[sundegrees[[i]] > 270 && sundegrees[[i]] < 300, sunsigns[[i]] = 10];
  If[sundegrees[[i]] > 300 && sundegrees[[i]] < 330, sunsigns[[i]] = 11];
  If[sundegrees[[i]] > 330 && sundegrees[[i]] < 360, sunsigns[[i]] = 12];];
```

In[158]:=
```
remove = Position[sunsigns, 0];
```

In[159]:=
```
Length[remove]
```

Out[159]=
```
564
```

Remove players with sun sign on cusp

In[160]:=
```
editedsunsigns = Delete[sunsigns, remove];
```

In[161]:=
```
editedheights = Delete[heights, remove];
```

In[162]:=
```
editedweights = Delete[weights, remove];
```

Just heights

In[ ]:=
```
editedheightinches =
  Table[editedheights[[i, 1]] * 12 + editedheights[[i, 2]], {i, 1, Length[editedheights]}];
```

In[ ]:=
```
analysisdata = Transpose[{editedsunsigns, editedheightinches}];
```

In[ ]:=
```
Length[analysisdata]
```

Out[ ]=
```
17066
```

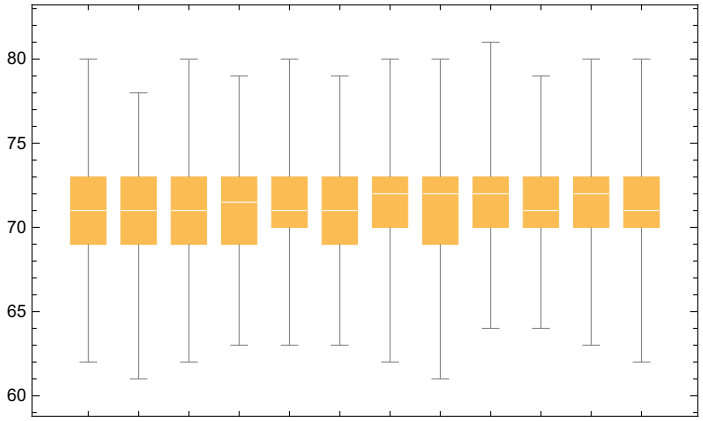
In[ ]:=
```
gathereddata = SortBy[GatherBy[analysisdata, First], First];
```

Box Whisker Chart

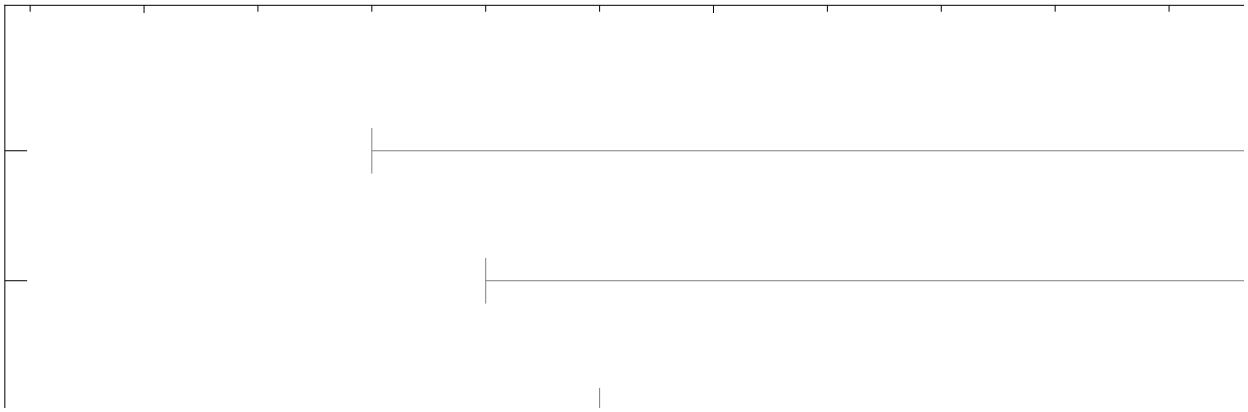
In[ ]:=
```
plotdata = Table[gathereddata[[i]][[All, 2]], {i, 1, 12}];
```

In[ ]:=
```
BoxWhiskerChart[plotdata]
```

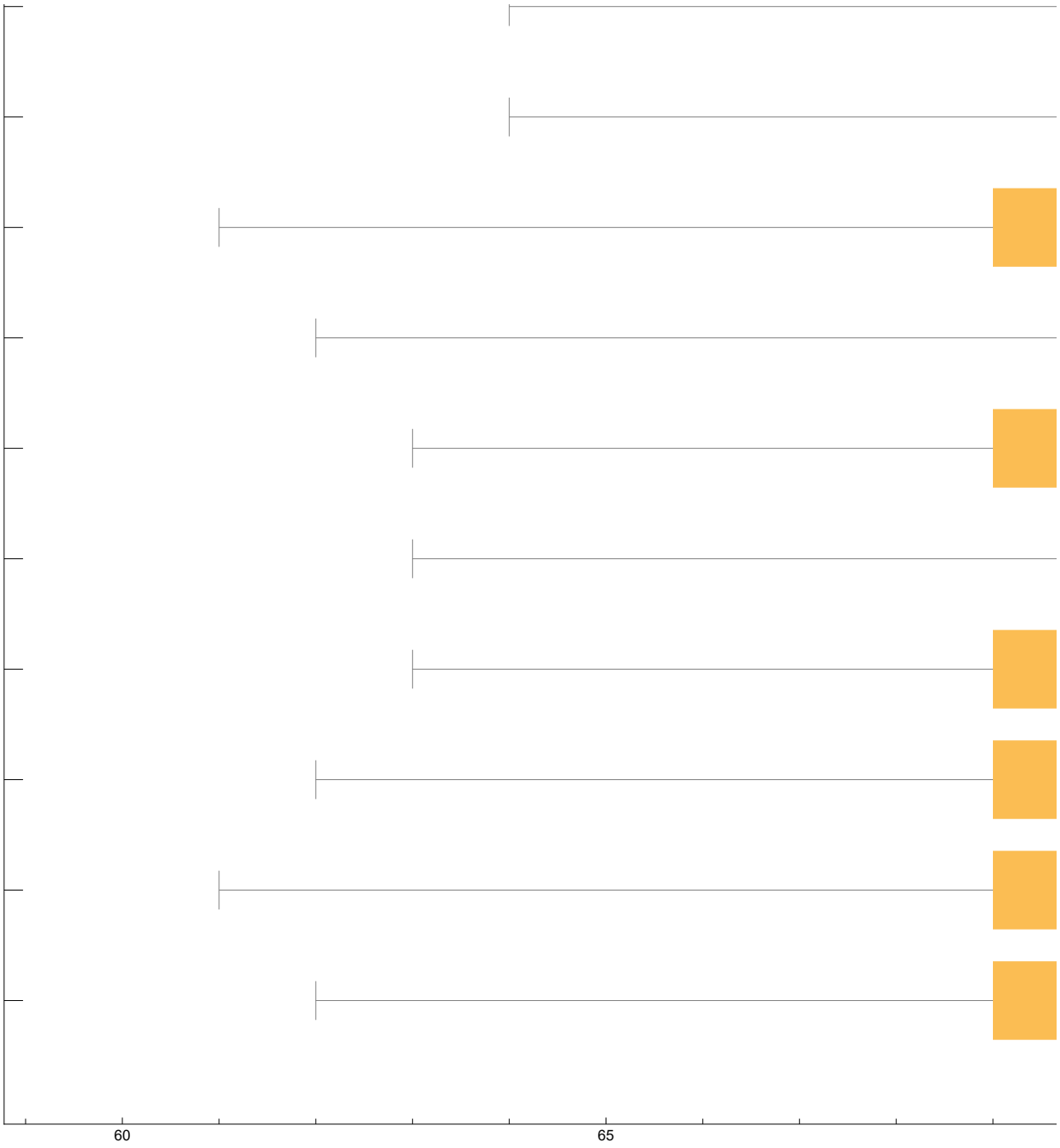
Out[ ]=



In[ ]:=
```
BoxWhiskerChart[plotdata,
 (*{"Diamond",{"MeanDiamond",1,Pink}},"*)"Notched", BarOrigin → Left]
```

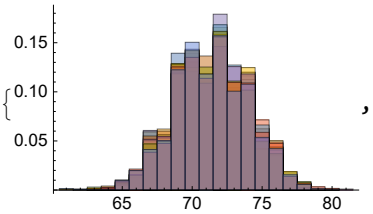


Out[ ]=

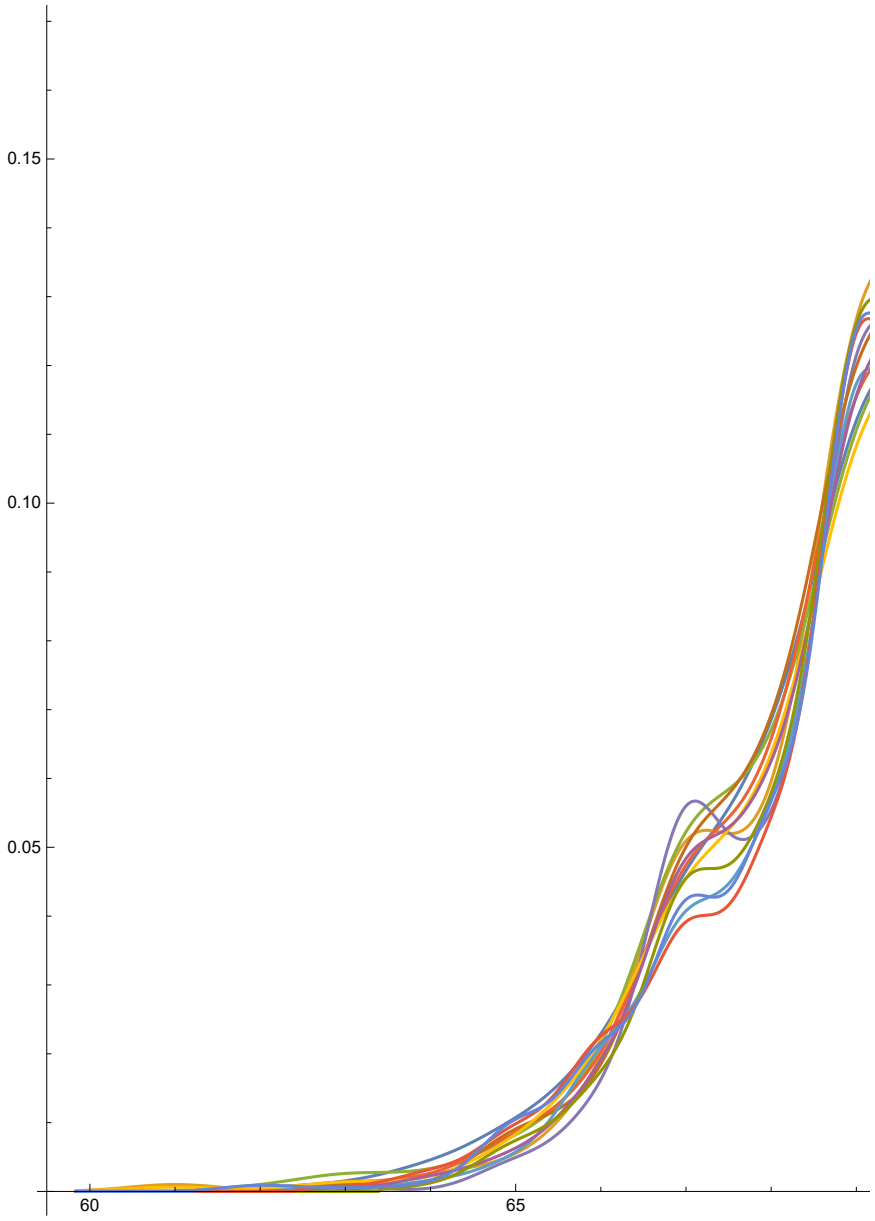


In[ ]:=
```
{Histogram[plotdata, Automatic, "PDF"], SmoothHistogram[plotdata]}
```

Out[ ]=



,




```
analysisdata
```

In[ ]:=
```
LocationEquivalenceTest[plotdata]
```

Out[ ]=
```
0.388979
```

ANOVA

In[ ]:=
```
Needs["ANOVA`"]
```

In[ ]:=
```
ANOVA[analysisdata, PostTests → Bonferroni]
```

General: $0.000664885^{8527.}$ is too small to represent as a normalized machine number; precision may be lost.

Out[ ]=
```
{ANOVA → 
          DF      SumOfSq     MeanSq     FRatio     PValue
Model     11      78.2185     7.11078    1.00726    0.43665 ,
Error     17054   120393.     7.0595
Total     17065   120471.

                All          71.3805
                Model[1]     71.3517
                Model[2]     71.32
                Model[3]     71.3215
                Model[4]     71.4146
                Model[5]     71.346
 CellMeans → {  Model[6]     71.2673 , PostTests → {Model → Bonferroni  {}}}
                Model[7]     71.519
                Model[8]     71.4288
                Model[9]     71.3974
                Model[10]    71.3594
                Model[11]    71.4774
                Model[12]    71.3697
```

Just weights

In[ ]:=
```
analysisdata = Transpose[{editedsunsigns, editedweights}];
```

In[ ]:=
```
Length[analysisdata]
```

Out[ ]=
```
17066
```

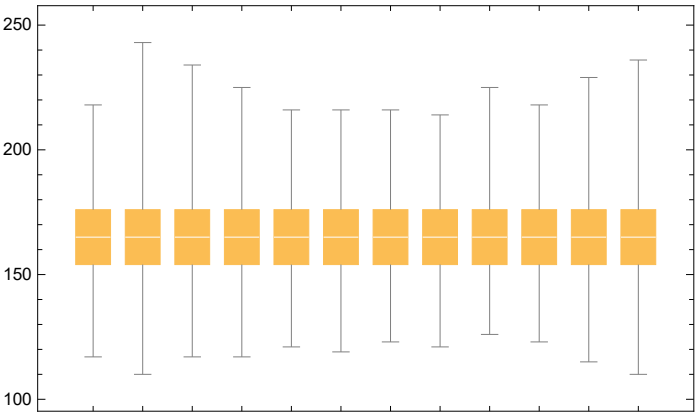
In[ ]:=
```
gathereddata = SortBy[GatherBy[analysisdata, First], First];
```

Box Whisker Chart

In[ ]:=
```
plotdata = Table[gathereddata[[i]][[All, 2]], {i, 1, 12}];
```

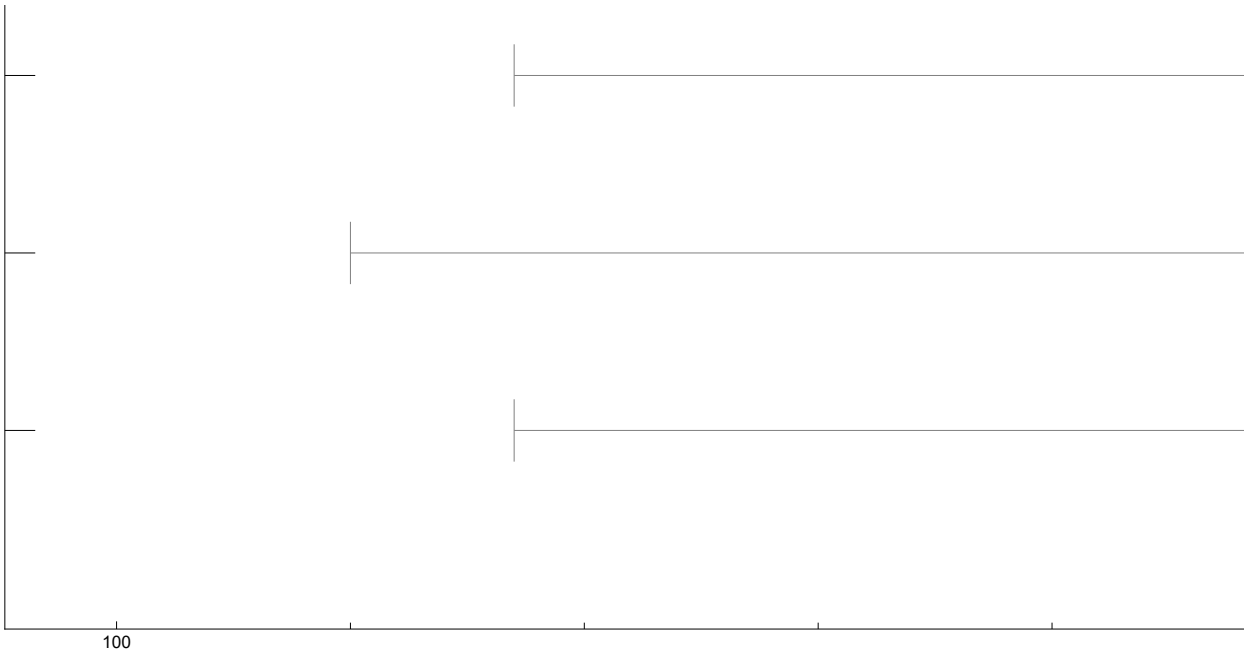
In[ ]:=
```
BoxWhiskerChart[plotdata]
```

Out[ ]=



In[ ]:=
```
BoxWhiskerChart[plotdata, {"Diamond", {"MeanDiamond", 1, Pink}}, BarOrigin → Left]
```

Out[ ]=



In[ ]:= **plotdata**

Out[ ]= { ··· 1 ··· }

large output | **show less** | **show more** | **show all** | **set size limit...**

analysisdata

In[ ]:= **LocationEquivalenceTest[plotdata]**

Out[ ]= 0.185874

ANOVA

In[ ]:= **Needs["ANOVA`"]**

In[•]:=
```
ANOVA[analysisdata, PostTests → Bonferroni]
```

General: 0.000664885$^{8527.}$ is too small to represent as a normalized machine number; precision may be lost.

Out[•]=
```
{ANOVA → 
          DF      SumOfSq          MeanSq     FRatio    PValue
Model     11      3407.07          309.734    1.2752    0.231606
Error     17 054  4.14225 × 10^6   242.89
Total     17 065  4.14566 × 10^6

                All          166.074
                Model[1]     166.073
                Model[2]     165.883
                Model[3]     165.449
                Model[4]     166.007
                Model[5]     165.888
CellMeans → {   Model[6]     165.507, PostTests → {Model → Bonferroni  {}}}
                Model[7]     166.894
                Model[8]     166.156
                Model[9]     166.12
                Model[10]    165.582
                Model[11]    166.844
                Model[12]    166.222
```

Calculate BMI from https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html#InterpretedAdults

Formula: weight (lb) / [height (in)]^2 x 703
Calculate BMI by dividing weight in pounds (lbs) by height in inches (in) squared and multiplying by a conversion factor of 703.

Example: Weight = 150 lbs, Height = 5’5″ (65″)
Calculation: [150 ÷ (65)^2] x 703 = 24.96

In[163]:=
```
editedheightinches =
  Table[editedheights[[i, 1]] * 12 + editedheights[[i, 2]], {i, 1, Length[editedheights]}];
```

In[164]:=
```
bmi = N[Table[(editedweights[[i]] / (editedheightinches[[i]]^2)) * 703,
    {i, 1, Length[editedweights]}]];
```

In[165]:=
```
analysisdata = Transpose[{editedsunsigns, bmi}];
```

In[166]:=
```
Length[analysisdata]
```

Out[166]=
```
17 066
```

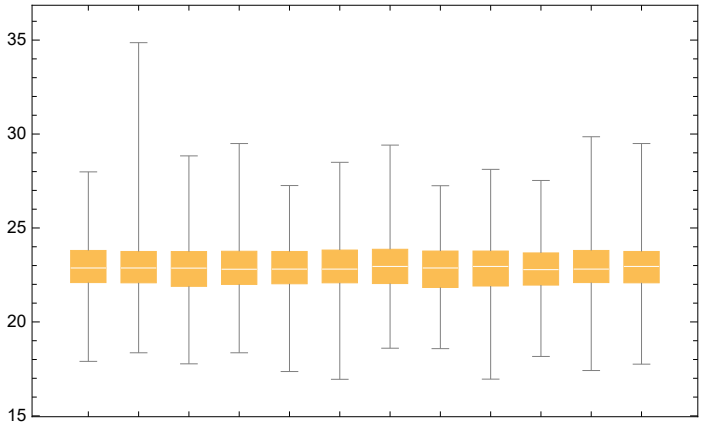
In[167]:=
```
gathereddata = SortBy[GatherBy[analysisdata, First], First];
```

Box Whisker Chart

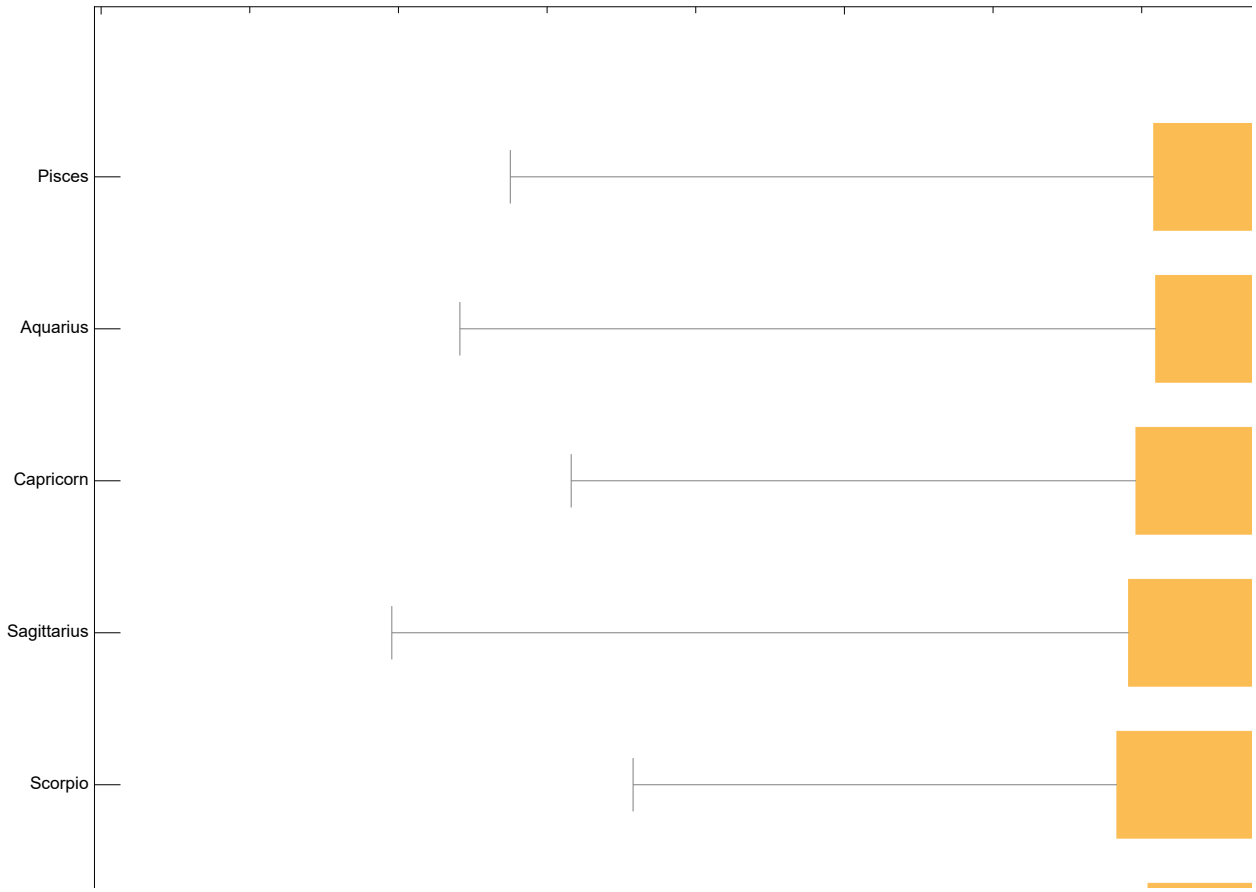
In[168]:=
```
plotdata = Table[gathereddata[[i]][[All, 2]], {i, 1, 12}];
```

In[169]:=
```
BoxWhiskerChart[plotdata]
```

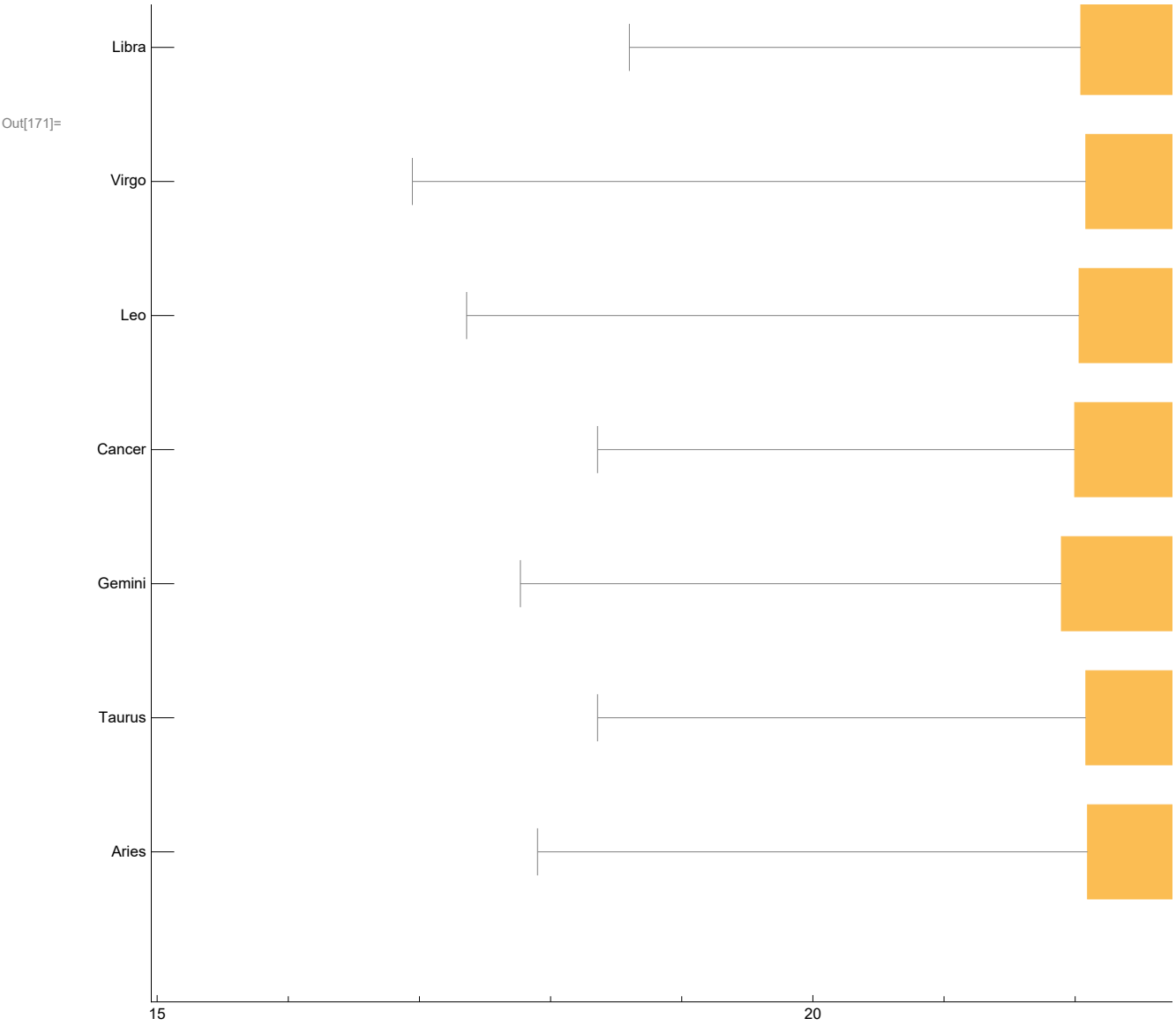
Out[169]=



In[171]:=
```
BoxWhiskerChart[plotdata, {"Diamond", {"MeanDiamond", 1, Pink}},
 BarOrigin → Left, ChartLabels → {"Aries", "Taurus", "Gemini", "Cancer", "Leo",
   "Virgo", "Libra", "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pisces"},
 ChartLabels → "BMI for soccer players with different Sun signs"]
```



Out[171]=



ANOVA

In[ ]:= **Needs["ANOVA`"]**

In[ ]:= **ANOVA[analysisdata, PostTests → Bonferroni]**

... General: 0.000664885$^{8527.}$ is too small to represent as a normalized machine number; precision may be lost.

Out[ ]=
```
{ANOVA → 
          DF      SumOfSq    MeanSq     FRatio     PValue
Model     11      15.4194    1.40176    0.717242   0.72313
Error     17 054  33 330.    1.95438
Total     17 065  33 345.4

          All          22.8877
          Model[1]     22.9063
          Model[2]     22.8977
          Model[3]     22.8343
          Model[4]     22.8588
          Model[5]     22.8864
CellMeans → Model[6]   22.8847, PostTests → {Model → Bonferroni  {}}}
          Model[7]     22.9149
          Model[8]     22.8664
          Model[9]     22.8808
          Model[10]    22.8368
          Model[11]    22.9323
          Model[12]    22.9121
```

Neural net to recognize faces as sun signs

In[ ]:= **Position[data[[1]], "Photo"]**

Out[ ]= {{5}}

In[ ]:= **imageURLs = data[[2 ;; All, 5]]**

(Dialog) In[ ]:= **i**

(Dialog) Out[ ]= 17 194

In[ ]:= **images = Table[Import[imageURLs[[i]]], {i, 1, Length[imageURLs]}];**

... FetchURL: The request to URL https://cdn.sofifa.org/players/4/19/230481.png was not successful. The server returned the HTTP status code 404 ("Not Found").

... FetchURL: The request to URL https://cdn.sofifa.org/players/4/19/230375.png was not successful. The server returned the HTTP status code 404 ("Not Found").

... FetchURL: The request to URL https://cdn.sofifa.org/players/4/19/230294.png was not successful. The server returned the HTTP status code 404 ("Not Found").

... General: Further output of FetchURL::httperr will be suppressed during this calculation.

In[ ]:= **images = Import["images.m"];**

Out[ ]= $Aborted

In[ ]:= **remove2 = Position[Table[ImageQ[images[[i]]], {i, 1, Length[images]}], False];**

In[ ]:= **remove3 = Union[remove, remove2];**

```
(*Export["images.m",images]*)
```

Out[ ]= images.m

```
editedimages = Delete[images, remove3];
```

```
editedsunsigns3 = Delete[sunsigns, remove3];
```

```
netdata = Table[editedimages[[i]] -> editedsunsigns3[[i]], {i, 1, Length[editedimages]}];
```

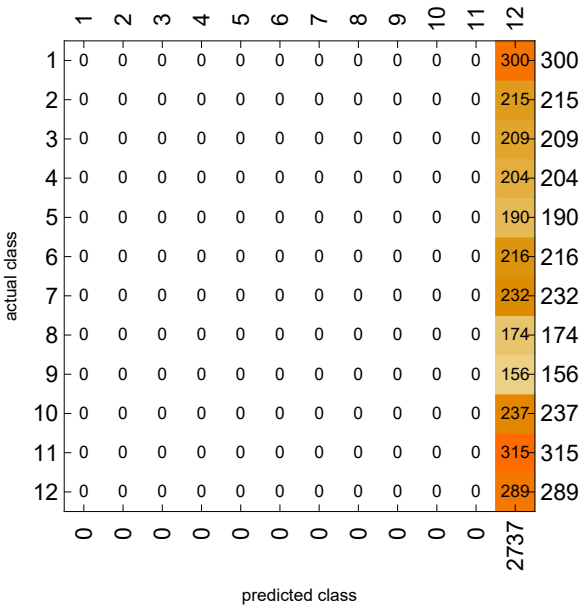
```
testindices = RandomSample[Range[Length[netdata]], Round[0.2 * Length[netdata]]];
```

```
trainindices = Complement[Range[Length[netdata]], testindices];
```

```
c = Classify[netdata[[trainindices]], PerformanceGoal → "Quality", TargetDevice → "GPU"];
```

```
ClassifierMeasurements[c, netdata[[testindices]], "ConfusionMatrixPlot"]
```

Out[ ]=

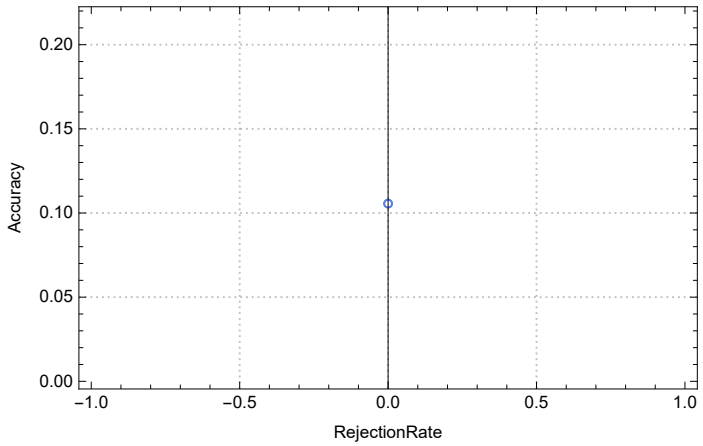



```
Length[remove3]
```

Out[ ]= 3944

In[ ]:=
```
ClassifierMeasurements[c, netdata[[testindices]], "AccuracyRejectionPlot"]
```

Out[ ]=



In[ ]:=
```
ClassifierMeasurements[c, netdata[[testindices]], "Accuracy"]
```

Out[ ]=
```
$Aborted
```

In[ ]:=
```
trainingNet = NetGraph[<|"lenet" → lenet, "loss" → CrossEntropyLossLayer["Index"]|>,
  {NetPort["Input"] → "lenet" → NetPort["loss", "Input"],
   NetPort["Target"] → NetPort["loss", "Target"]}]
```

NetGraph: lenet is not a layer, a net, or a valid specification for one.

Out[ ]=
```
$Failed
```

```
mnistEncoder = NetEncoder[{"Image", {48, 48}, "ColorSpace" → "CMYK"}]
```

Out[ ]=

NetEncoder

Type:	Image
Image size:	{48, 48}
Color space:	RGB
Color channels:	3
Mean image:	None
Variance image:	None
Output:	3–tensor (size: 3 × 48 × 48)

In[ ]:=
```
mnistDecoder = NetDecoder[{"Class", Range[1, 12]}]
```

Out[ ]=

NetDecoder

Type:	Class
Labels:	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Input depth:	1
Dimensions:	12

```
In[•]:= uninitializedLenet =
         NetChain[{ConvolutionLayer[20, 5], ElementwiseLayer[Ramp], PoolingLayer[2, 2],
           ConvolutionLayer[50, 5], ElementwiseLayer[Ramp], PoolingLayer[2, 2], FlattenLayer[],
           LinearLayer[500], ElementwiseLayer[Ramp], LinearLayer[12], SoftmaxLayer[]},
          "Input" → mnistEncoder, "Output" → mnistDecoder]
```

Out[•]= NetChain[uninitialized | Input port: image; Output port: class; Number of layers: 11]

```
In[•]:= lenet = NetInitialize[uninitializedLenet]
```

Out[•]= NetChain[Input port: image; Output port: class; Number of layers: 11]

```
In[•]:= Length[testindices]
```

Out[•]= 2737

```
In[•]:= trainedlenet = NetTrain[(*NetModel["LeNet"]*)lenet,
          netdata[[trainindices]], ValidationSet → netdata[[testindices[[1 ;; 2500]]]],
          MaxTrainingRounds → 100, TargetDevice → "GPU"]
```

Out[•]= NetChain[Input port: image; Output port: class; Number of layers: 11]

```
In[•]:= measurements = ClassifierMeasurements[trainedlenet, netdata[[testindices[[2501 ;; All]]]]]
```

Out[•]= ClassifierMeasurementsObject[Classifier: Net; Number of test examples: 237]

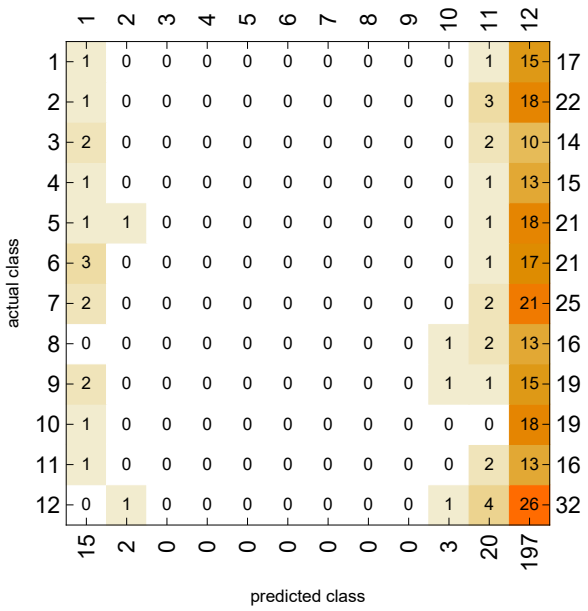
Data not in notebook; Store now »

```
In[•]:= measurements["Accuracy"]
```

Out[•]= 0.122363

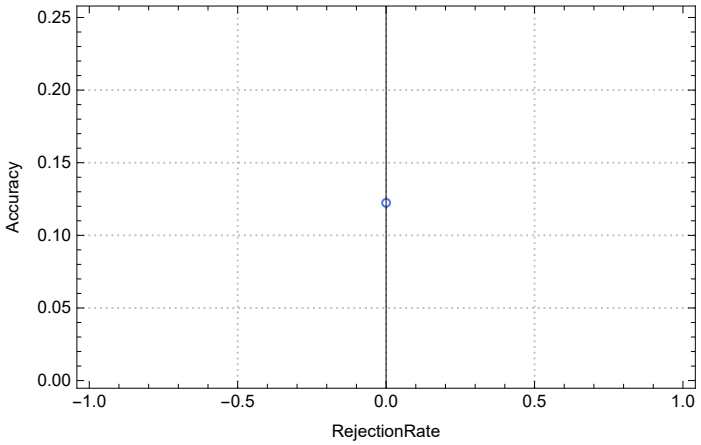
In[ ]:=
```
measurements["ConfusionMatrixPlot"]
```

Out[ ]=



In[ ]:=
```
measurements["AccuracyRejectionPlot"]
```

Out[ ]=



Repeat everything but use Sidereal placements

Calculate Tropical sun degrees and signs

```
(*Export["sundegrees.m",sundegrees]*)
```

(Dialog) Out[ ]= sundegrees.m

In[6]:=
```
sundegrees = Import["sundegrees.m"];
```

In[7]:=
```
sidsundegrees = Round[Mod[sundegrees - 23.7, 360]];
```

```
(*cusplist={};
For[i=1,i≤Length[sundegrees],i++,
 If[sundegrees[[i]]==0||sundegrees[[i]]==30||
   sundegrees[[i]]==60||sundegrees[[i]]==90||sundegrees[[i]]==120||
   sundegrees[[i]]==150||sundegrees[[i]]==180||sundegrees[[i]]==210||
   sundegrees[[i]]==240||sundegrees[[i]]==270||sundegrees[[i]]==300||
   sundegrees[[i]]==330||sundegrees[[i]]==360,AppendTo[cusplist,i]]];*)
```

```
(*cusplist*)
```

In[ ]:=
```
{13, 38, 47}
```

Out[ ]= {13, 38, 47}

In[8]:=
```
sidsunsigns = ConstantArray[0, Length[sidsundegrees]];
For[i = 1, i ≤ Length[sidsundegrees], i++,
  If[sidsundegrees[[i]] > 0 && sidsundegrees[[i]] < 30, sidsunsigns[[i]] = 1];
  If[sidsundegrees[[i]] > 30 && sidsundegrees[[i]] < 60, sidsunsigns[[i]] = 2];
  If[sidsundegrees[[i]] > 60 && sidsundegrees[[i]] < 90, sidsunsigns[[i]] = 3];
  If[sidsundegrees[[i]] > 90 && sidsundegrees[[i]] < 120, sidsunsigns[[i]] = 4];
  If[sidsundegrees[[i]] > 120 && sidsundegrees[[i]] < 150, sidsunsigns[[i]] = 5];
  If[sidsundegrees[[i]] > 150 && sidsundegrees[[i]] < 180, sidsunsigns[[i]] = 6];
  If[sidsundegrees[[i]] > 180 && sidsundegrees[[i]] < 210, sidsunsigns[[i]] = 7];
  If[sidsundegrees[[i]] > 210 && sidsundegrees[[i]] < 240, sidsunsigns[[i]] = 8];
  If[sidsundegrees[[i]] > 240 && sidsundegrees[[i]] < 270, sidsunsigns[[i]] = 9];
  If[sidsundegrees[[i]] > 270 && sidsundegrees[[i]] < 300, sidsunsigns[[i]] = 10];
  If[sidsundegrees[[i]] > 300 && sidsundegrees[[i]] < 330, sidsunsigns[[i]] = 11];
  If[sidsundegrees[[i]] > 330 && sidsundegrees[[i]] < 360, sidsunsigns[[i]] = 12];];
```

In[10]:=
```
remove = Position[sidsunsigns, 0];
```

In[ ]:=
```
Length[remove]
```

Out[ ]= 535

Remove players with sun sign on cusp

```
In[ ]:= editedsunsigns = Delete[sidsunsigns, remove]
```

```
In[ ]:= editedheights = Delete[heights, remove]
```

```
In[ ]:= editedweights = Delete[weights, remove]
```

Calculate BMI from https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html#InterpretedAdults

Formula: weight (lb) / [height (in)]^2 x 703
Calculate BMI by dividing weight in pounds (lbs) by height in inches (in) squared and multiplying by a conversion factor of 703.

Example: Weight = 150 lbs, Height = 5'5″ (65″)
Calculation: [150 ÷ (65)^2] x 703 = 24.96

```
In[ ]:= editedheightinches =
  Table[editedheights[[i, 1]] * 12 + editedheights[[i, 2]], {i, 1, Length[editedheights]}];
```

```
In[ ]:= bmi = N[Table[(editedweights[[i]] / (editedheightinches[[i]]^2)) * 703,
    {i, 1, Length[editedweights]}]];
```

```
In[ ]:= analysisdata = Transpose[{editedsunsigns, bmi}];
```

```
In[ ]:= Length[analysisdata]
```

```
Out[ ]= 17095
```

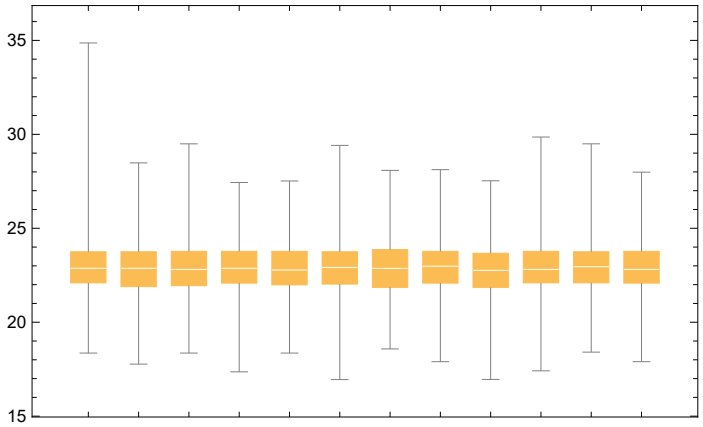
```
In[ ]:= gathereddata = SortBy[GatherBy[analysisdata, First], First];
```

Box Whisker Chart

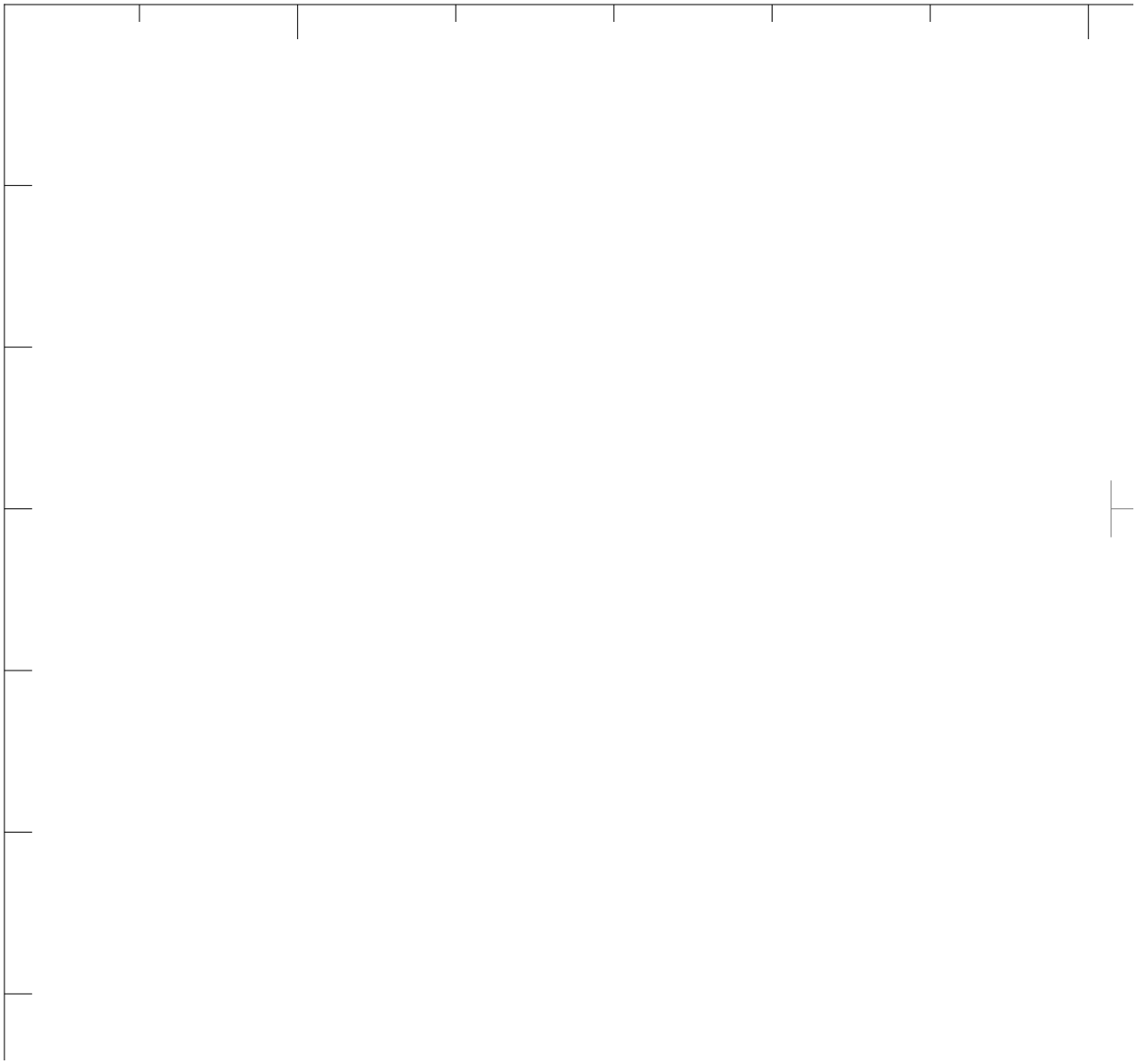
```
In[ ]:= plotdata = Table[gathereddata[[i]][[All, 2]], {i, 1, 12}];
```

In[ ]:=
```
BoxWhiskerChart[plotdata]
```

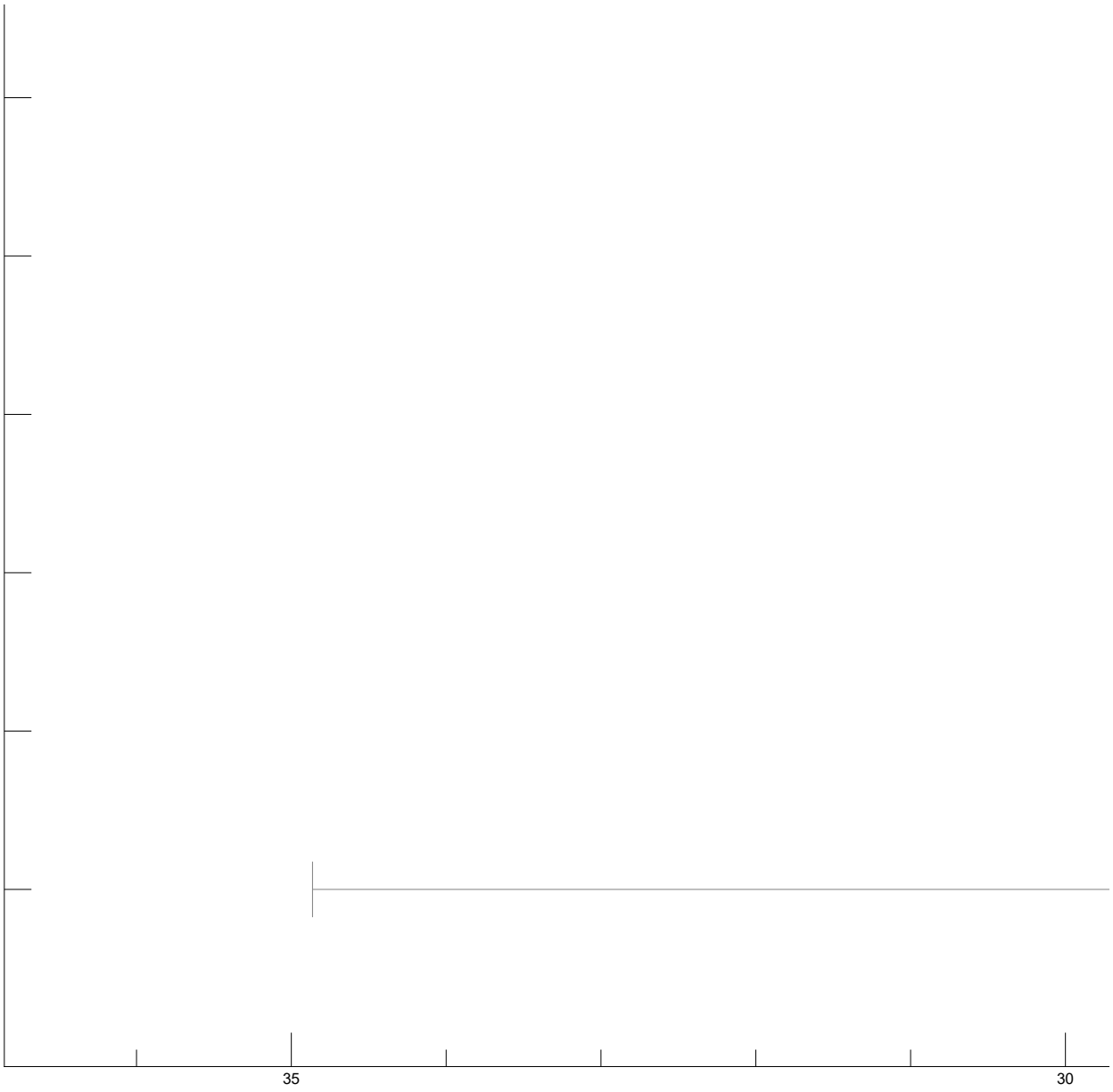
Out[ ]=



In[ ]:=
```
BoxWhiskerChart[plotdata, {"Diamond", {"MeanDiamond", 1, Pink}}, BarOrigin → Right]
```



Out[ ]=



ANOVA

In[ ]:= **Needs["ANOVA`"]**

```
In[ ]:= ANOVA[analysisdata, PostTests → Bonferroni]
```

... General: 0.000663757^8541.5 is too small to represent as a normalized machine number; precision may be lost.

```
Out[ ]= {ANOVA → 
          Model    11       24.0954    2.19049    1.11757    0.342096
          Error    17083    33483.5    1.96005
          Total    17094    33507.6

                 All          22.8852
                 Model[1]     22.8961
                 Model[2]     22.8441
                 Model[3]     22.8556
                 Model[4]     22.9243
                 Model[5]     22.8483
        CellMeans → Model[6]  22.8865, PostTests → {Model → Bonferroni  {}}}
                 Model[7]     22.8848
                 Model[8]     22.9238
                 Model[9]     22.8018
                 Model[10]    22.9283
                 Model[11]    22.9239
                 Model[12]    22.8764
```

Neural net to recognize faces as sun signs

```
In[ ]:= Position[data[[1]], "Photo"]
Out[ ]= {{5}}

In[ ]:= imageURLs = data[[2 ;; All, 5]]

(Dialog) In[ ]:= i
(Dialog) Out[ ]= 17194

In[ ]:= images = Table[Import[imageURLs[[i]]], {i, 1, Length[imageURLs]}];

In[4]:= images = Import["images.m"];

In[5]:= remove2 = Position[Table[ImageQ[images[[i]]], {i, 1, Length[images]}], False];

In[ ]:= Length[remove2]
Out[ ]= 3480

In[11]:= remove3 = Union[remove, remove2];

         (*Export["images.m",images]*)

(Dialog) In[49]:= Length[remove3]
(Dialog) Out[49]= 3922

In[ ]:= ColorConvert[images[[1]], "RGB"]
Out[ ]=
```

In[12]:=
```
editedimages = Delete[images, remove3];

Table[Export["\\soccerpics\\" <> ToString[i] <> ".jpg",
   ColorConvert[editedimages[[i]], "RGB"]], {i, 1, Length[editedimages]}];

editedimages3 = Table[ColorConvert[editedimages[[i]], "RGB"], {i, 1, Length[editedimages]}]
```

In[51]:=
```
Length[editedimages]
```

Out[51]= 13 708

In[13]:=
```
editedsidsunsigns3 = Delete[sidsunsigns, remove3];
```

(Dialog) In[51]:=
```
i
```

(Dialog) Out[51]= 1811

```
netdata = Table[Import["\soccerpics\\" <> ToString[i] <> ".jpg"] -> editedsidsunsigns3[[i]],
   {i, 1, Length[editedimages]}];
```

In[55]:=
```
ClearAll[images, editedimages]
```

In[56]:=
```
Union[Values[netdata]]
```

Out[56]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

In[57]:=
```
testindices = RandomSample[Range[Length[netdata]], Round[0.2 * Length[netdata]]];
```

In[58]:=
```
trainindices = Complement[Range[Length[netdata]], testindices];
```

In[17]:=
```
Length[testindices]
```

Out[17]= 2742

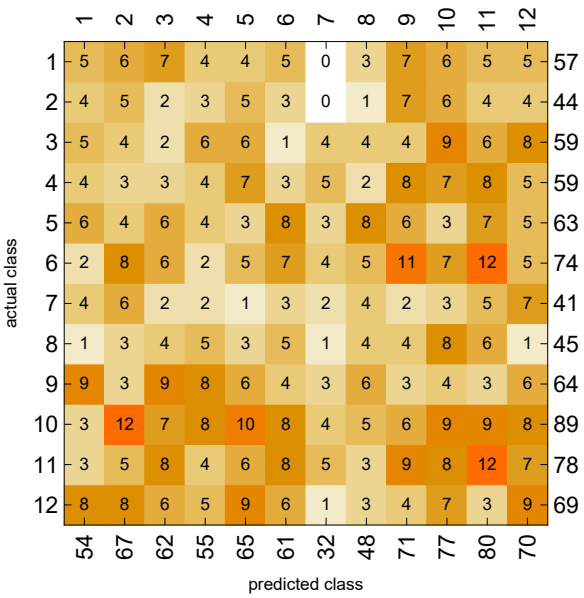
In[59]:=
```
train = netdata[[trainindices]];
```

In[60]:=
```
validation = netdata[[trainindices[[1 ;; 2000]]]];
```

In[ ]:=
```
c = Classify[train, TargetDevice → "GPU",
   ValidationSet → validation, PerformanceGoal → "Quality"];
```

In[ ]:= **ClassifierMeasurements[c, netdata[[testindices[[2001 ;; All]]]], "ConfusionMatrixPlot"]**

Out[ ]=



In[ ]:= **ClassifierMeasurements[c, netdata[[testindices[[2001 ;; All]]]], "Accuracy"]**

Out[ ]= 0.0876011

(Dialog) In[ ]:= **c**

(Dialog) Out[ ]= ClassifierFunction[Input type: Image
Number of classes: 13
Method: DecisionTree
Number of training examples: 10 949]

In[ ]:= **ImageDimensions[editedimages[[1]]]**

Out[ ]= {48, 48}

In[ ]:= **Length[remove3]**

Out[ ]= 3944

In[ ]:=
```
ImageData[editedimages[[1]]]
```

Out[ ]=
```
{{{0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0.439216, 0.388235, 0.407843, 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0.560784, 0.517647, 0.54902, 0.},
  {0.815686, 0.788235, 0.847059, 0.}, {0.827451, 0.8, 0.858824, 0.}, {0., 0., 0., 0.},
  {0.835294, 0.807843, 0.866667, 0.}, {0.839216, 0.811765, 0.870588, 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.},
  {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}, {0., 0., 0., 0.}},
 {<<1>>}, <<44>>, {<<1>>}, {{0.611765, 0.588235, 0.635294, 0.},
  {0.25098, 0.262745, 0.568627, 0.}, {0., 0., 0., 0.},
  {0.403922, 0.392157, 0.635294, 0.0901961}, {0.211765, 0.235294, 0.517647, 0.627451},
  {0.172549, 0.211765, 0.498039, 1.}, <<36>>,
  {0.211765, 0.258824, 0.607843, 0.992157}, {0.235294, 0.278431, 0.666667, 1.},
  {0.25098, 0.286275, 0.67451, 1.}, {0.262745, 0.286275, 0.658824, 0.733333},
  {0.345098, 0.341176, 0.694118, 0.235294}, {0., 0., 0., 0.}}}
```

large output | **show less** | **show more** | **show all** | **set size limit...**

In[61]:=
```
mnistEncoder = NetEncoder[{"Image", {48, 48}, "ColorSpace" → "RGB"}]
```

Out[61]=
NetEncoder[

Type:	Image
Image size:	{48, 48}
Color space:	RGB
Color channels:	3
Mean image:	None
Variance image:	None
Output:	3–tensor (size: 3 × 48 × 48)

]

In[62]:=
```
mnistDecoder = NetDecoder[{"Class", Range[1, 12]}]
```

Out[62]=
NetDecoder[

Type:	Class
Labels:	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Input depth:	1
Dimensions:	12

]

In[138]:=
```
uninitializedLenet =
 NetChain[{ConvolutionLayer[20, 5], ElementwiseLayer[Ramp], PoolingLayer[5, 5],
   ConvolutionLayer[100, 5], ElementwiseLayer[Ramp], PoolingLayer[4, 4], FlattenLayer[],
   LinearLayer[500], ElementwiseLayer[Ramp], LinearLayer[12], SoftmaxLayer[]},
  "Input" → mnistEncoder, "Output" → mnistDecoder]
```

Out[138]= NetChain[uninitialized | Input port: image; Output port: class; Number of layers: 11]

In[139]:=
```
lenet = NetInitialize[uninitializedLenet]
```

Out[139]= NetChain[Input port: image; Output port: class; Number of layers: 11]

In[65]:=
```
Length[testindices]
```

Out[65]= 2742

In[144]:=
```
trainedlenet = NetTrain[(*NetModel["LeNet"]*)
  lenet, train, ValidationSet → netdata[[trainindices[[1 ;; 2000]]]],
  BatchSize → 500, MaxTrainingRounds → 2000, TargetDevice → "GPU"]
```

Out[144]= NetChain[Input port: image; Output port: class; Number of layers: 11]

In[35]:=
```
Internal`$LastInternalFailure
```

Out[35]=

MXNetError

```
[20:15:04]: Check failed: assign(&dattr, (*vec)[i]) Incompatible attr in node  at 0-th output: ex
```

In[148]:=
```
measurements = ClassifierMeasurements[trainedlenet, netdata[[testindices[[2001 ;; All]]]]]
```

Out[148]= ClassifierMeasurementsObject[Classifier: Net; Number of test examples: 742]

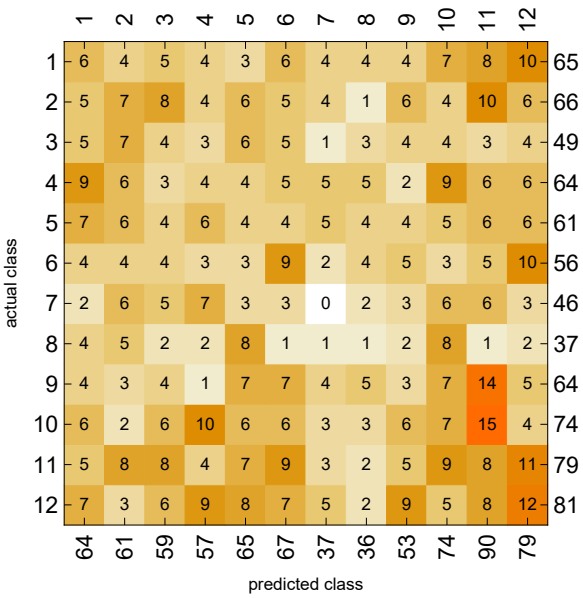
Data not in notebook; Store now »

In[149]:=
```
measurements["Accuracy"]
```

Out[149]= 0.0876011

In[150]:=
```
measurements["ConfusionMatrixPlot"]
```

Out[150]=

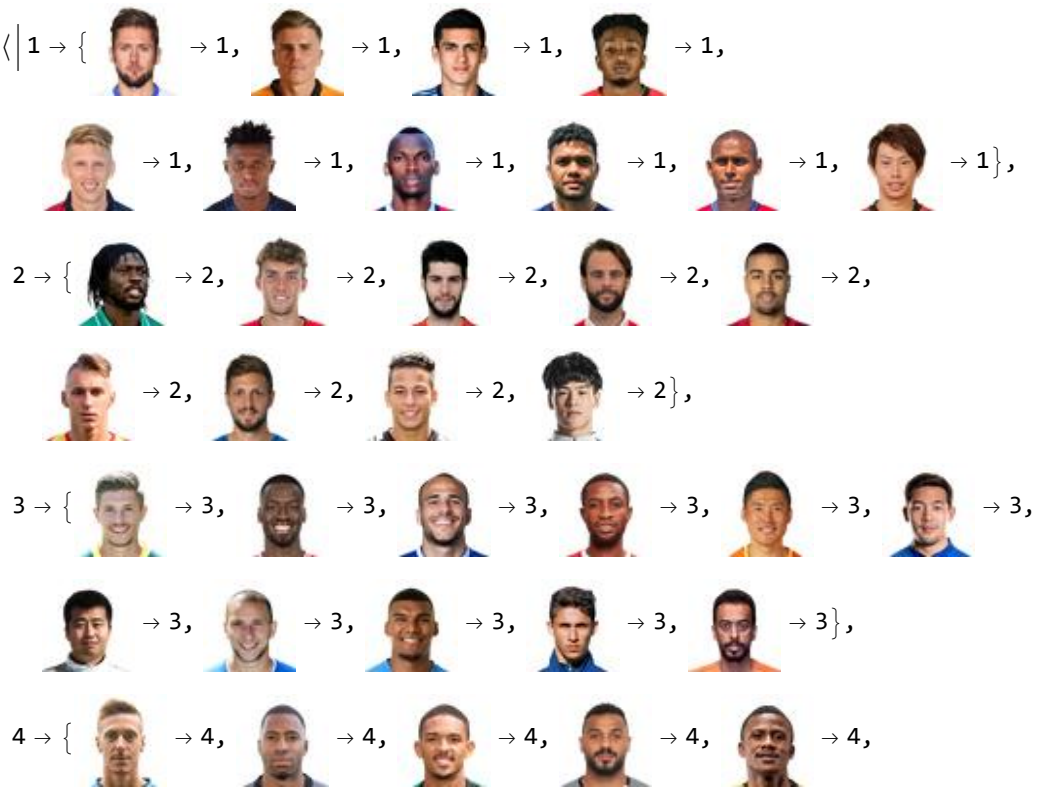


In[96]:=
```
N[191 / Length[netdata[[testindices[[1001 ;; All]]]]]]
```

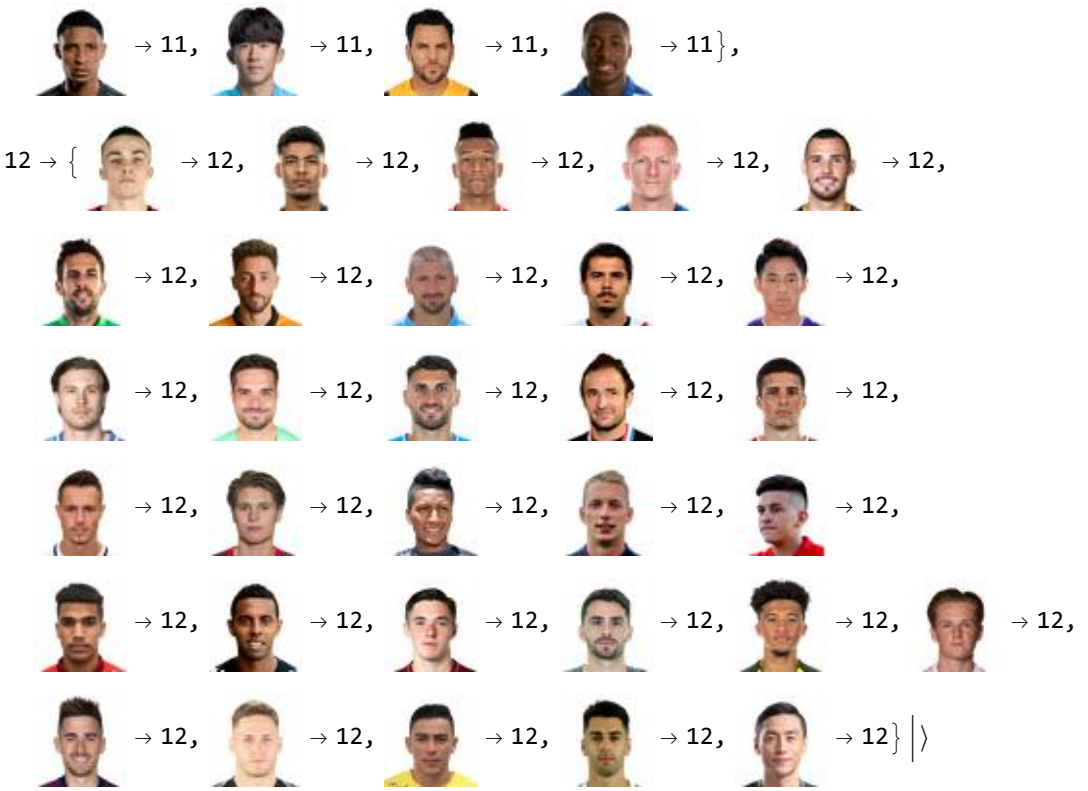
Out[96]=
```
0.109644
```

In[90]:=
```
measurements["TruePositiveExamples"]
```

Out[90]=




→ 4, → 4, → 4, → 4, → 4},
5 → { → 5, → 5, → 5, → 5, → 5, → 5, → 5, → 5,
→ 5, → 5, → 5, → 5, → 5, → 5, → 5,
→ 5, → 5, → 5, → 5, → 5, → 5, → 5},
6 → { → 6, → 6, → 6, → 6, → 6, → 6},
7 → { → 7, → 7},
8 → { → 8, → 8, → 8, → 8, → 8, → 8},
9 → { → 9, → 9},
10 → { → 10, → 10, → 10, → 10, → 10, → 10,
→ 10, → 10, → 10, → 10, → 10, → 10, → 10,
→ 10, → 10, → 10, → 10, → 10, → 10, → 10,
→ 10, → 10, → 10, → 10, → 10},
11 → { → 11, → 11, → 11, → 11, → 11,
→ 11, → 11, → 11, → 11, → 11,



(Dialog) In[ ]:= `Length[testindices]`

(Dialog) Out[ ]= 2737