In[5]:=
```
baseDirectory = "~/Documents/osrm/";
mxDirectory = FileNameJoin[{ParentDirectory[NotebookDirectory[]], "MX"}];
```

Setting up OSRM

Download relevant OSM data from https://download.geofabrik.de.

Preprocessing

This example shows how to preprocessing (using CH):

osrm-extract north-america-latest.osm.pbf -p /usr/local/share/osrm/profiles/car.lua
osrm-contract north-america-latest.osrm

To start the server, run:

osrm-routed north-america-latest.osrm --max-table-size=1000000000

Use the constantSpeed.lua profile for the extraction step to get constant speed variants.

Interface functions

The address of the routing server:

In[7]:=
```
routingBase = "http://0.0.0.0:5000/";
```

Formats coordinates appropriately for OSRM endpoints:

In[8]:=
```
coordinateOsrmRequest[ps_] :=
 StringJoin@Flatten@Riffle[Riffle[Reverse@#, ","] & /@ Map[ToString, ps, {2}], ";"]
coordinateOsrmRequest[ps : {__GeoPosition}] :=
 coordinateOsrmRequest[ps[[All, 1, ;; 2]]]
coordinateOsrmRequest[gp_GeoPosition] := coordinateOsrmRequest[Thread[gp]]
```

Computes distance matrices, returning both the distance matrix and time matrix (in units of meters and seconds)

In[11]:=
```
osrmDistanceMatrices[ps_, d_ : Quantity[Infinity, "Meters"]] :=
 Enclose@With[{r = Confirm[ImportString[ToString[Confirm@URLRead[URLBuild[{
             routingBase,
             "table", "v1", "driving",
             coordinateOsrmRequest[ps]},
            {"annotations" → "distance,duration", "generate_hints" → False}
           ], "Body"], CharacterEncoding → "UTF8"], "RawJSON"]]},
   With[{indices =
      Position[Quantity[#, "Meters"] & /@ Confirm[r["sources"]][[All, "distance"]],
        _?(LessThan[d]), {1}, Heads → False][[All, 1]]},
    <|"TravelDistances" → QuantityArray[Confirm[r["distances"]][[
         indices, indices]] /. Null → Missing[], "Meters"],
     "TravelDurations" → QuantityArray[Confirm[r["durations"]][[
         indices, indices]] /. Null → Missing[], "Seconds"],
     "Locations" → GeoPosition[Reverse /@ Confirm[r["sources"]][[
         indices, "location"]]]|>]]
```

Gridded population estimates

Available at https://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse.

We use the 2020 version of “UN WPP-Adjusted Population Count, v4.11”, available at https://sedac.ciesin.columbia.edu/data/set/gpw-v4-population-count-adjusted-to-2015-unwpp-country-totals-rev11/data-download.

Import the maximum resolution map:

In[12]:=
```
populationMap = Ramp@Import[FileNameJoin[{baseDirectory,
       "gpw-v4-population-count-adjusted-to-2015-unwpp-country-totals-rev11_2020_
         30_sec_tif",
       "gpw_v4_population_count_adjusted_to_2015_unwpp_country_totals_rev11_2020_
         30_sec.tif"}]];
```

Define a function for converting between grid coordinates and lat/long:

In[13]:=
```
coordPixel[{lat_, long_}] :=
 {Rescale[long, {-180, 180}, {0, ImageDimensions[populationMap][[1]]}],
  Rescale[lat, {-90, 90}, {0, ImageDimensions[populationMap][[2]]}]}
coordPixel[GeoPosition[{lat_, long_}]] := coordPixel[{lat, long}]
```

In[15]:=
```
pixelCoord[{x_, y_}] :=
 GeoPosition[{Rescale[y, {0, ImageDimensions[populationMap][[2]]}, {-90, 90}],
   Rescale[x, {0, ImageDimensions[populationMap][[1]]}, {-180, 180}]}]
```

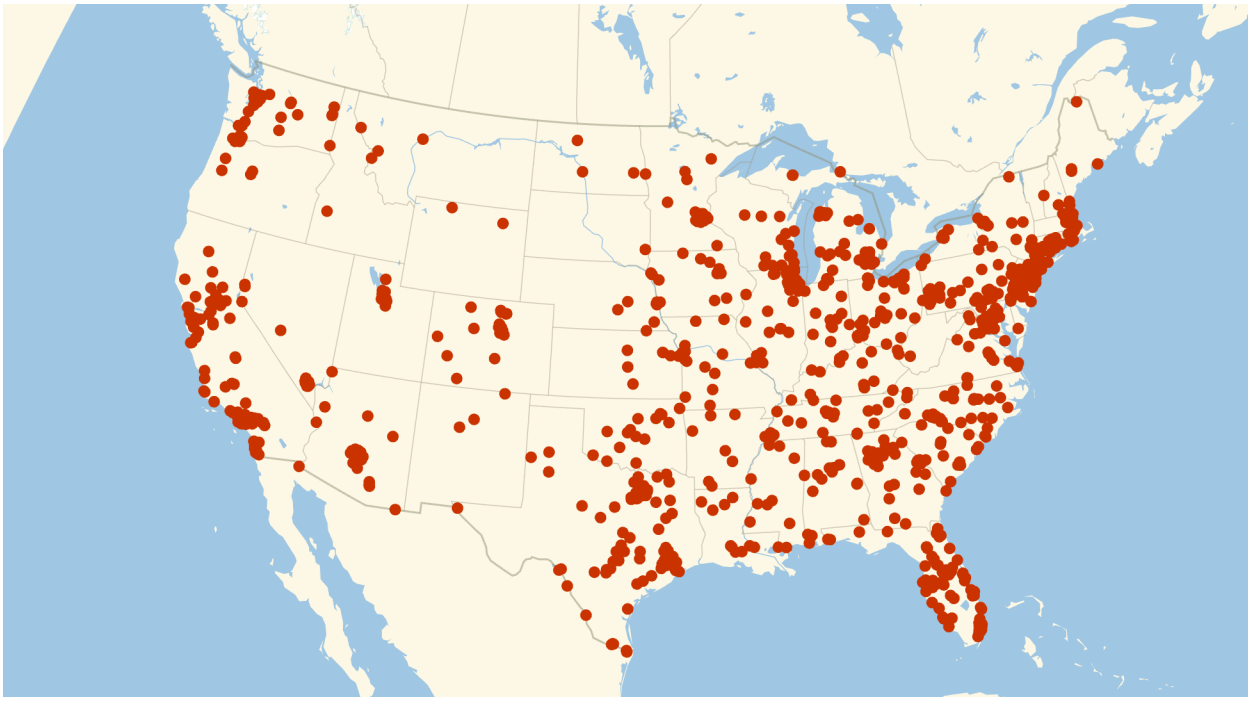
Define a function for randomly sampling population weighted points:

In[16]:=
```
populationWeightedSample[p_, n_] :=
 Module[{pixelBounds, coordBounds, dims,
   mask, localPopMap, borderFuzz, targetPixels, probs, pixels},
  pixelBounds = {Floor[#1], Ceiling[#2]} & @@ coordPixel /@ GeoBoundingBox[p];
  coordBounds =
   {pixelCoord[pixelBounds[[1]]][[1]], pixelCoord[pixelBounds[[2]]][[1]]};
  dims = -Subtract @@ pixelBounds;
  mask = ImageReflect[ColorNegate[Binarize[
      Rasterize[Graphics[Style[p /. GeoPosition → Identity, Antialiasing → False],
        PlotRange → Transpose[coordBounds]](*,ImageSize→Reverse[dims]*),
       RasterSize → Reverse[dims]]]], Top → Right];
  (*Alternate version using GeoGraphics. More robust but slower.*)
  (*mask=ColorNegate@Binarize[Rasterize[
       GeoGraphics[{GeoStyling[Opacity[1]],Style[p,Antialiasing→False]},
        GeoRange→Transpose[coordBounds],
        GeoProjection→"Equirectangular",GeoBackground→White],
       ImageSize→-Subtract@@pixelBounds,RasterSize→-Subtract@@pixelBounds]];*)
  localPopMap = ImageTrim[populationMap, pixelBounds + {1, -1}];
  (*Border fuzz makes sure that coastal
   cities get counted by adding fuzz around oceans*)
  borderFuzz = With[{i = Binarize[localPopMap, {0, 0}]}, i - Erosion[i, 1]];
  targetPixels =
   PixelValuePositions[Binarize[mask + Dilation[mask, 1] * borderFuzz, 0.5], 1];
  probs = PixelValue[localPopMap, targetPixels];
  If[Max[probs] > 0,
   pixels = RandomChoice[probs → targetPixels, n];
   (*This adds noise so that points between pixels can be chosen*)
   pixels += RandomReal[{0, 1}, If[ListQ[n], Append[n, 2], {n, 2}]];
   GeoPosition@Map[Reverse[MapThread[Rescale,
        {#, Thread[{1, dims}], Reverse[Transpose[coordBounds]]}]] &,
     pixels, {If[ListQ[n], Length[n], 1]}],
   GeoPosition[{}]]
 ]
populationWeightedSample[GeoDisk[c_, r_], n_] :=
 populationWeightedSample[Polygon[GeoDestination[c, {r, Range[0, 360, 1]}]], n]
```

Randomly pick points in the United States (only plotting the contiguous United States):

In[ ]:=
```
GeoListPlot[populationWeightedSample[
  Continental US states ADMINISTRATIVE DIVISIONS["Polygon"], 1000], ...]
```

Out[ ]=



Route Efficiency

General

Define a function for generating population-weighted random points and calculate the associated distance/time matrices:

In[18]:=
```
generateRandomTables[poly_, n_, d_ : Quantity[5, "Kilometers"]] :=
 With[{r = osrmDistanceMatrices[populationWeightedSample[poly, n], d]},
  If[FailureQ[r],
   <|"Locations" → 0,
    "GeoDistances" → {},
    "TravelDistances" → {},
    "TravelDurations" → {}|>,
   <|"Locations" → r["Locations"],
    "GeoDistances" → DistanceMatrix[r["Locations"]],
    "TravelDistances" → r["TravelDistances"],
    "TravelDurations" → r["TravelDurations"]|>]]
```

Define functions for computing the distance and travel efficiency:

In[19]:=
```
flattenDistanceMatrix[mat_] :=
 If[mat === {} || mat === {{}} || mat === GeoPosition[{}], {},
  Replace[Catenate[MapIndexed[Delete, mat]], _Missing | 0 | 0. → Indeterminate, {1}]]
```

In[20]:=
```
distanceEfficiency[data_] :=
 flattenDistanceMatrix[QuantityMagnitude[data["GeoDistances"], "Meters"]] /
  flattenDistanceMatrix[QuantityMagnitude[data["TravelDistances"], "Meters"]]
```

In[21]:=
```
timeEfficiency[data_] :=
 QuantityArray[
  flattenDistanceMatrix[QuantityMagnitude[data["GeoDistances"], "Meters"]] /
   flattenDistanceMatrix[QuantityMagnitude[data["TravelDurations"], "Seconds"]],
  "Meters" / "Seconds"]
```

Gather basic statistics given distance matrices:

In[22]:=
```
matricesBasicStatistics[r_] :=
 <|"DistanceEfficiency" →
   With[{e = DeleteCases[distanceEfficiency[r], Indeterminate]},
    If[Length[e] === 0,
     <|"Mean" → Missing[], "Median" → Missing[], "StandardDeviation" → Missing[]|>,
     <|"Mean" → Mean[e], "Median" → Median[e],
      "StandardDeviation" → StandardDeviation[e]|>]],
  "TimeEfficiency" → With[{e = QuantityArray[DeleteCases[
        QuantityMagnitude[timeEfficiency[r], "Meters" / "Seconds"],
        Indeterminate], "Meters" / "Seconds"]},
    If[Length[e] === 0,
     <|"Mean" → Missing[], "Median" → Missing[], "StandardDeviation" → Missing[]|>,
     <|"Mean" → Mean[e], "Median" → Median[e],
      "StandardDeviation" → StandardDeviation[e]|>]]|>
```

Local sampling

Define a function for sampling points on a triangular lattice:

In[23]:=
```
triangularGridPoints[{{xmin_, xmax_}, {ymin_, ymax_}}, count_] :=
 With[{w = xmax - xmin, h = ymax - ymin,
   d = 3 ^ (3 / 4) * Sqrt[count] / Sqrt[2 * (xmax - xmin) * (ymax - ymin)]},
  Catenate@Table[({3 x / 2, Sqrt[3] y} + {0, Sqrt[3] / 2 * Boole[EvenQ[x]]}) / d +
     {xmin, ymin}, {x, 0, 2 * w / 3 * d}, {y, 0, h / Sqrt[3] * d}]]
triangularGridPoints[r_?ConstantRegionQ, count_] :=
 With[{ps = triangularGridPoints[RegionBounds[r],
     count * Times @@ Subtract @@@ RegionBounds[r] / Area[r]]},
  Pick[ps, RegionMember[r, ps]]]
```

The latter definition will generate approximately **count** points on a triangular lattice within the region **r**.

For each of **n** sample points on a triangular lattice in **region**, compute the distance and travel efficiency of a disk with radius 50 miles:

In[25]:=
```
generateLocalResults[region_, proj_, n_, rad_ : Quantity[50, "Miles"]] :=
 GeneralUtilities`MonitoredMap[
  With[{r = generateRandomTables[GeoDisk[#, rad], 100]},
    Join[
     <|"Location" → #,
      "SamplePoints" → Length[Thread[r["Locations"]]]|>,
     matricesBasicStatistics[r]]] &,
  Thread[GeoPosition[GeoGridPosition[triangularGridPoints[region, n], proj]]]]
```

Analysis

Generating data

United States:

In[ ]:=
```
usData =
  generateRandomTables[ Continental US states ADMINISTRATIVE DIVISIONS ["Polygon"], 2000];

Export[FileNameJoin[{mxDirectory, "usData.wxf"}], usData, OverwriteTarget → False];
```

In[26]:=
```
usData = Import[FileNameJoin[{mxDirectory, "usData.wxf"}]];
```

Europe:

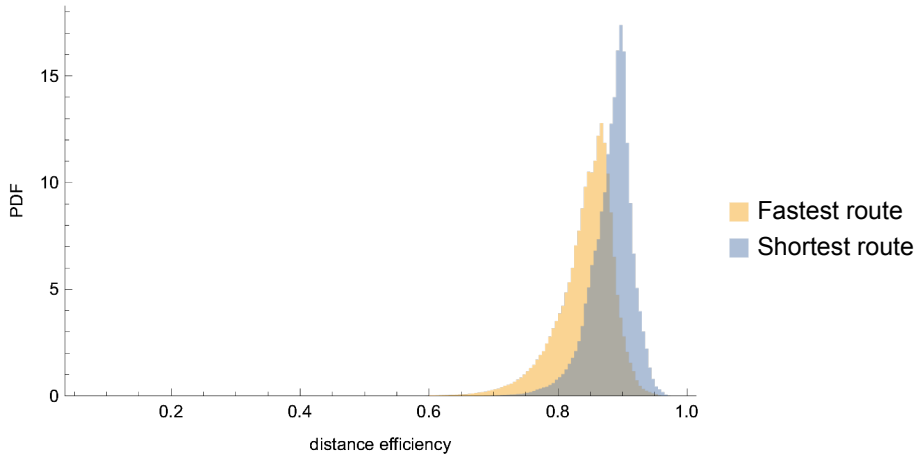
In[ ]:=
```
europeData = generateRandomTables[ Europe GEOGRAPHIC REGION ["Polygon"], 2000];

Export[FileNameJoin[{mxDirectory, "europeData.wxf"}],
  europeData, OverwriteTarget → False];
```

In[27]:=
```
europeData = Import[FileNameJoin[{mxDirectory, "europeData.wxf"}]];
```

Constant rate

In[ ]:=
```
Histogram[{distanceEfficiency[usData], distanceEfficiency[usConstantData]},
 {0.005}, "PDF", Frame → {{True, False}, {True, False}},
 FrameLabel → {"distance efficiency", "PDF"},
 ChartLegends → {"Fastest route", "Shortest route"}]
```

Out[ ]=

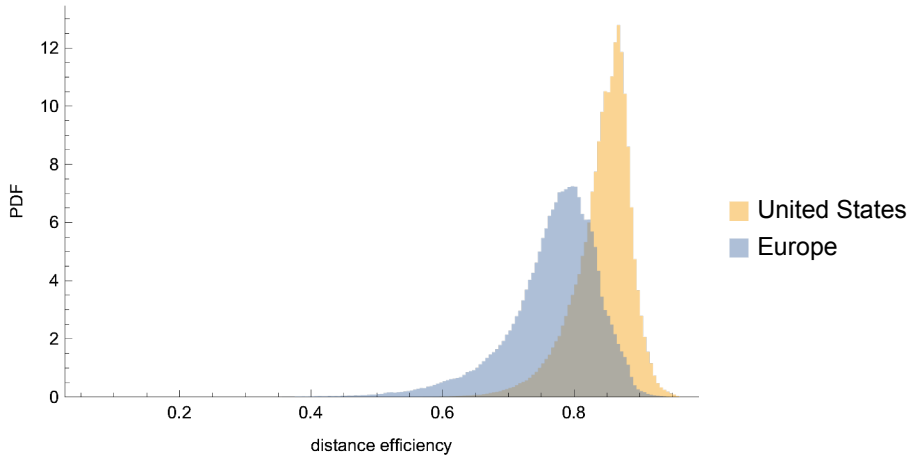


Basic analysis

Visualize the results:

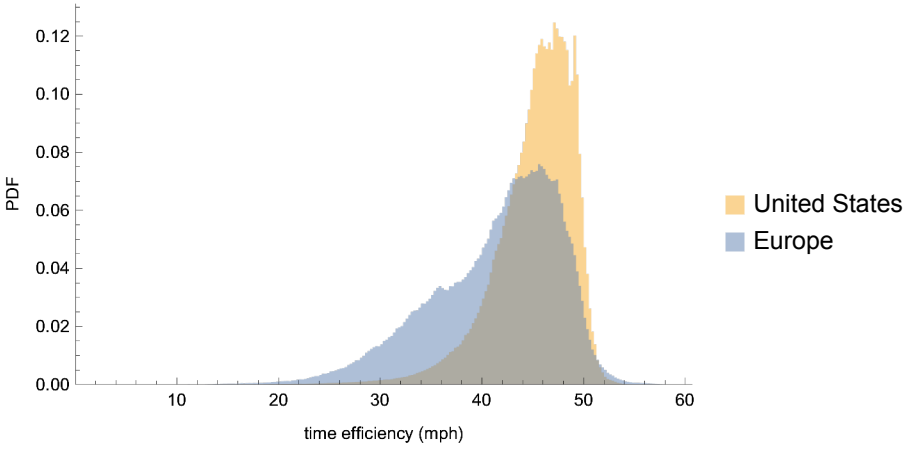
In[ ]:=
```
Histogram[{distanceEfficiency[usData], distanceEfficiency[europeData]},
 {0.005}, "PDF", Frame → {{True, False}, {True, False}},
 FrameLabel → {"distance efficiency", "PDF"},
 ChartLegends → {"United States", "Europe"}]
```

Out[ ]=



In[•]:=
```
Histogram[{
  QuantityMagnitude[timeEfficiency[usData], "Miles" / "Hours"],
  QuantityMagnitude[timeEfficiency[europeData], "Miles" / "Hours"]},
 {0.25}, "PDF", Frame → {{True, False}, {True, False}},
 FrameLabel → {"time efficiency (mph)", "PDF"},
 ChartLegends → {"United States", "Europe"}]
```

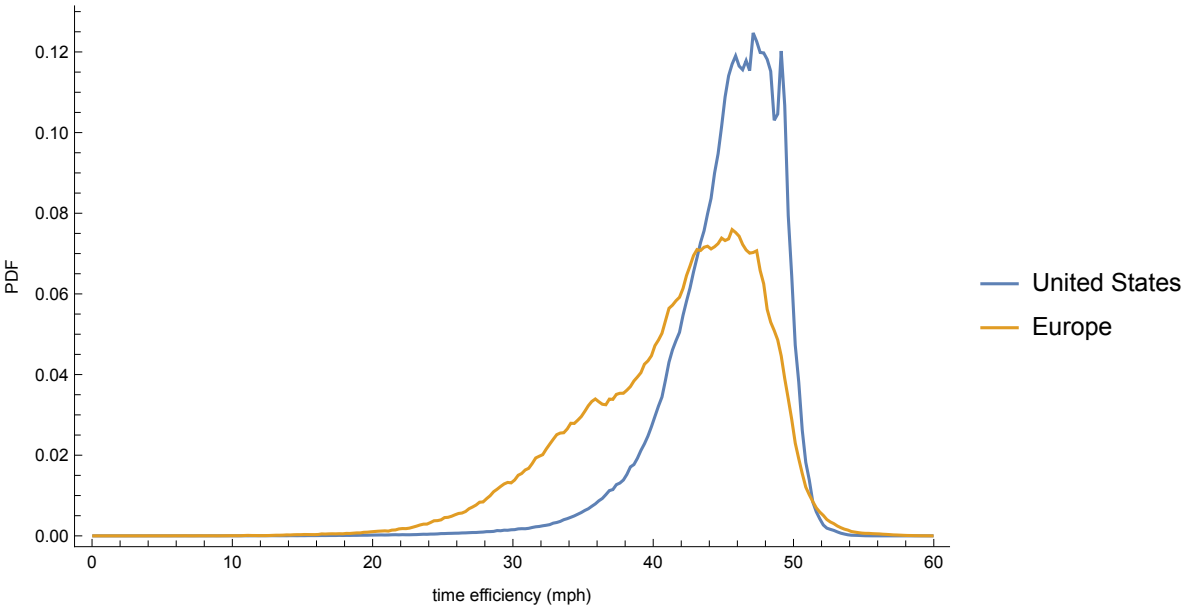
Out[•]=



In[•]:=
```
histList[d_, bspec_ : {0.5, 1, 0.001}] :=
 With[{hl = HistogramList[d, bspec, "PDF"]},
  Transpose[{BlockMap[Mean, hl[[1]], 2, 1], hl[[2]]}]]
```

In[ ]:=
```
ListLinePlot[{
  histList[
   QuantityMagnitude[timeEfficiency[usData], "Miles" / "Hours"], {0, 60, 0.25}],
  histList[QuantityMagnitude[timeEfficiency[europeData], "Miles" / "Hours"],
   {0, 60, 0.25}]}, Frame → {{True, False}, {True, False}},
 FrameLabel → {"time efficiency (mph)", "PDF"},
 PlotLegends → {"United States", "Europe"}]
```

Out[ ]=



Route efficiency vs distance

Look at route efficiency as a function of geodesic distance:

In[•]:=
```
ListLinePlot[{
  KeyValueMap[List, Around[Mean[#], StandardDeviation[#]] & /@ KeySort[GroupBy[
      Transpose[{flattenDistanceMatrix[QuantityMagnitude[usData[
            "TravelDistances"], "Kilometers"]], distanceEfficiency[usData]}],
      (Round[#[[1]], 10] &) → Last]]],
  KeyValueMap[List, Around[Mean[#], StandardDeviation[#]] & /@ KeySort[GroupBy[
      Transpose[{flattenDistanceMatrix[QuantityMagnitude[europeData[
            "TravelDistances"], "Kilometers"]], distanceEfficiency[europeData]}],
      (Round[#[[1]], 10] &) → Last]]]
 }, Frame → {{True, False}, {True, False}},
 FrameLabel → {"distance (kilometers)", "distance efficiency"},
 PlotLegends → {"United States", "Europe"},
 IntervalMarkers → "Bands", PlotRange → {All, {0.6, 1}}]
```

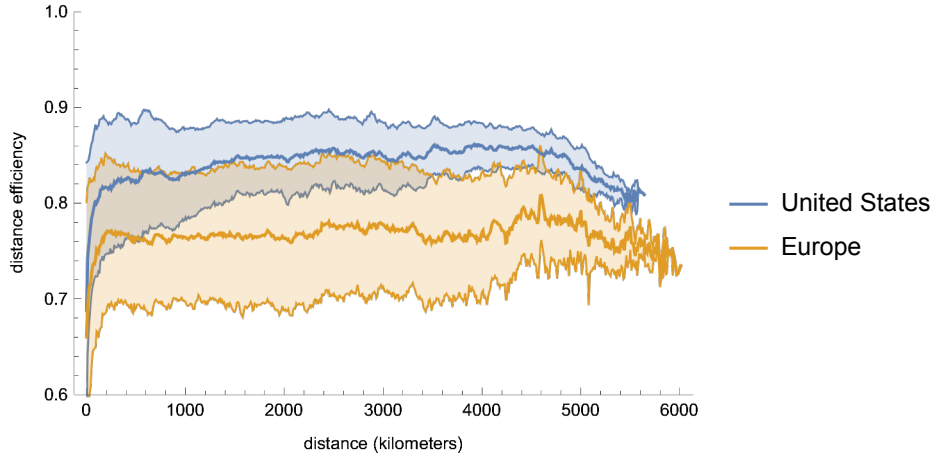
••• StandardDeviation: The argument {0.81302} should have at least two elements.

••• StandardDeviation: The argument {0.810248} should have at least two elements.

••• StandardDeviation: The argument {0.808832} should have at least two elements.

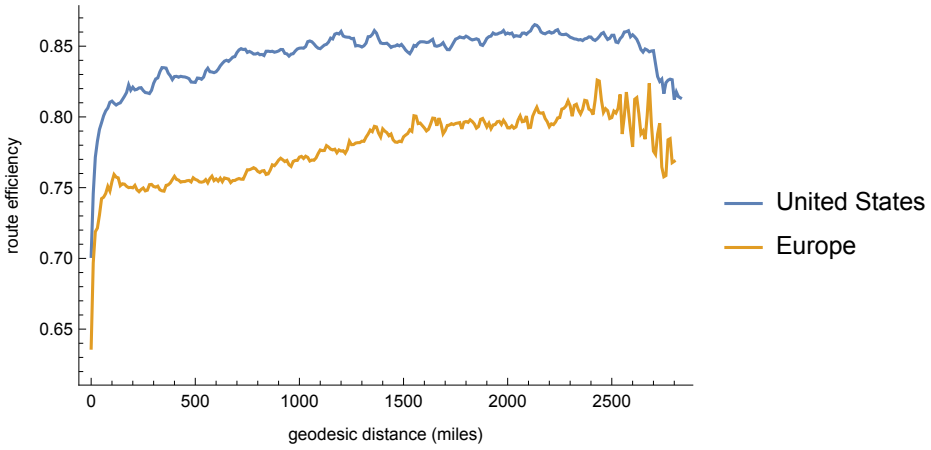
••• General: Further output of StandardDeviation::shlen will be suppressed during this calculation.

Out[•]=



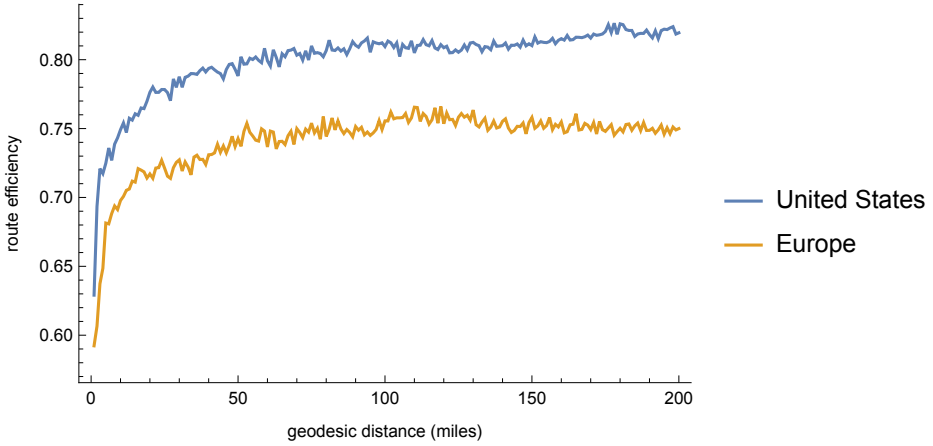
In[ ]:=
```
ListLinePlot[{
  KeyValueMap[List, Mean /@ KeySort[GroupBy[
      Transpose[{flattenDistanceMatrix[QuantityMagnitude[
          usData["GeoDistances"], "Miles"]], distanceEfficiency[usData]}],
      (Round[#[[1]], 10] &) → Last]]],
  KeyValueMap[List, Mean /@ KeySort[GroupBy[
      Transpose[{flattenDistanceMatrix[QuantityMagnitude[europeData["GeoDistances"]
          "Miles"]], distanceEfficiency[europeData]}],
      (Round[#[[1]], 10] &) → Last]]]
 }, Frame → {{True, False}, {True, False}},
 FrameLabel → {"geodesic distance (miles)", "route efficiency"},
 PlotLegends → {"United States", "Europe"}]
```

Out[ ]=



In[ ]:=
```
ListLinePlot[{
  MovingMap[Mean, Select[Transpose[
     {flattenDistanceMatrix[QuantityMagnitude[usData["GeoDistances"], "Miles"]],
      distanceEfficiency[usData]}], #[[1]] < 250 &], {1, Center, {0, 200, 1}}],
  MovingMap[Mean, Select[Transpose[{flattenDistanceMatrix[
       QuantityMagnitude[europeData["GeoDistances"], "Miles"]],
      distanceEfficiency[europeData]}], #[[1]] < 250 &], {1, Center, {0, 200, 1}}]},
 Frame → {{True, False}, {True, False}}, FrameLabel →
  {"geodesic distance (miles)", "route efficiency"},
 PlotLegends → {"United States", "Europe"}]
```

Out[ ]=




```
ListLinePlot[{
  MovingMap[Mean, Transpose[{flattenDistanceMatrix[
       QuantityMagnitude[usData["GeoDistances"], "Kilometers"]],
      distanceEfficiency[usData]}], {10, Center, {Automatic, Automatic, 10}}],
  MovingMap[Mean, Transpose[{flattenDistanceMatrix[
       QuantityMagnitude[europeData["GeoDistances"], "Kilometers"]],
      distanceEfficiency[europeData]}], {10, Center, {Automatic, Automatic, 10}}]
 }, Frame → {{True, False}, {True, False}}, FrameLabel →
  {"distance (kilometers)", "distance efficiency"},
 PlotLegends → {"United States", "Europe"}]
```

MovingMap: The window specification {10, Center, {Automatic, Automatic, 10}} is invalid.

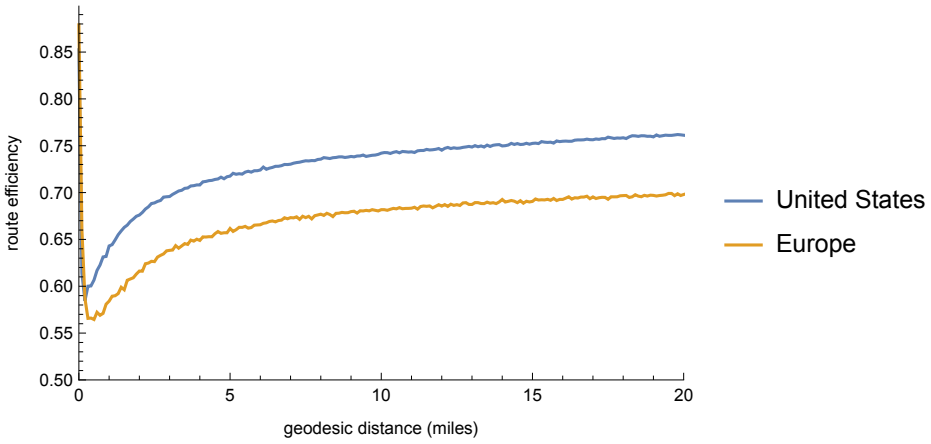
Out[ ]=
```
$Aborted
```

Very short routes:

In[ ]:=
```
usShortRangeTables = GeneralUtilities`MonitoredMap[
   generateRandomTables[GeoDisk[#, Quantity[25, "Miles"]], 100] &,
   Thread[populationWeightedSample[
     Continental US states ADMINISTRATIVE DIVISIONS ["Polygon"], 1000]]];
```

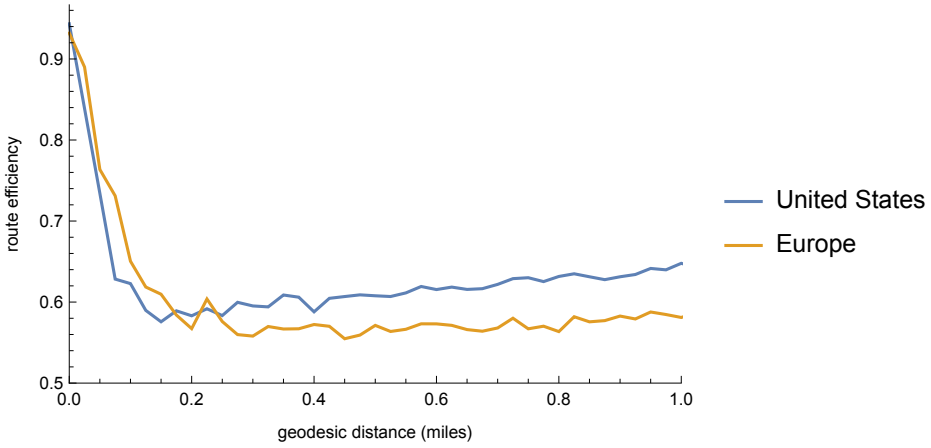
```
In[ ]:= Export[FileNameJoin[{mxDirectory, "usShortRangeTables.wxf"}],
          usShortRangeTables, OverwriteTarget → False];

In[36]:= usShortRangeTables = Import[FileNameJoin[{mxDirectory, "usShortRangeTables.wxf"}]];

In[ ]:= europeShortRangeTables = GeneralUtilities`MonitoredMap[
           generateRandomTables[GeoDisk[#, Quantity[25, "Miles"]], 100] &,
           Thread[populationWeightedSample[ Europe GEOGRAPHIC REGION ["Polygon"], 1000]]];

In[ ]:= Export[FileNameJoin[{mxDirectory, "europeShortRangeTables.wxf"}],
          europeShortRangeTables, OverwriteTarget → False];

In[35]:= europeShortRangeTables =
          Import[FileNameJoin[{mxDirectory, "europeShortRangeTables.wxf"}]];

In[ ]:= ListLinePlot[{
          KeyValueMap[List, Mean /@ KeySort[GroupBy[
              Catenate[Transpose[{flattenDistanceMatrix[QuantityMagnitude[#["GeoDistances"]
                      "Miles"]], distanceEfficiency[#]}] & /@ usShortRangeTables],
              (Round[#[[1]], 0.1] &) → Last]]],
          KeyValueMap[List, Mean /@ KeySort[GroupBy[
              Catenate[Transpose[{flattenDistanceMatrix[QuantityMagnitude[#["GeoDistances"]
                      "Miles"]], distanceEfficiency[#]}] & /@ europeShortRangeTables],
              (Round[#[[1]], 0.1] &) → Last]]]},
        Frame → {{True, False}, {True, False}}, PlotRange → {{0, 20}, {0.5, All}},
        FrameLabel → {"geodesic distance (miles)", "route efficiency"},
        PlotLegends → {"United States", "Europe"}]
```

Out[ ]=



```
In[•]:= ListLinePlot[{
          KeyValueMap[List, Mean /@ KeySort[GroupBy[
              Catenate[Transpose[{flattenDistanceMatrix[QuantityMagnitude[#["GeoDistances"]
                    "Miles"]], distanceEfficiency[#]}] & /@ usShortRangeTables],
              (Round[#[[1]], 0.025] &) → Last]]],
          KeyValueMap[List, Mean /@ KeySort[GroupBy[
              Catenate[Transpose[{flattenDistanceMatrix[QuantityMagnitude[#["GeoDistances"]
                    "Miles"]], distanceEfficiency[#]}] & /@ europeShortRangeTables],
              (Round[#[[1]], 0.025] &) → Last]]]},
        Frame → {{True, False}, {True, False}}, PlotRange → {{0, 1}, {0.5, All}},
        FrameLabel → {"geodesic distance (miles)", "route efficiency"},
        PlotLegends → {"United States", "Europe"}]
```

Out[•]=



State-by-state

For each country in Europe (excluding Russia), compute the travel efficiency:

```
stateData = generateRandomTables[#, 1000] & /@ EntityValue[
      Continental US states ADMINISTRATIVE DIVISIONS, "Polygon", "EntityAssociation"];
```

```
In[•]:= stateData =
          GeneralUtilities`MonitoredMap[generateRandomTables[#, 1000] &, EntityValue[
            Continental US states ADMINISTRATIVE DIVISIONS, "Polygon", "EntityAssociation"]];

        Export[FileNameJoin[{mxDirectory, "stateData.wxf"}],
          stateData, OverwriteTarget → False];
```

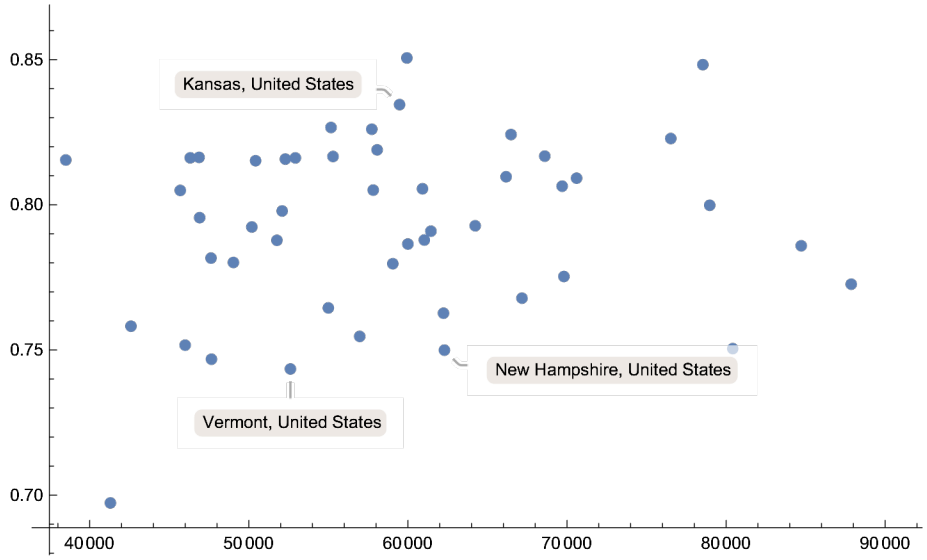
```
In[31]:= stateData = Import[FileNameJoin[{mxDirectory, "stateData.wxf"}]];
```

```
In[•]:= stateStats = matricesBasicStatistics /@ stateData;
```

```
In[1]:= stateStats = ⟨|...|⟩ ;
```

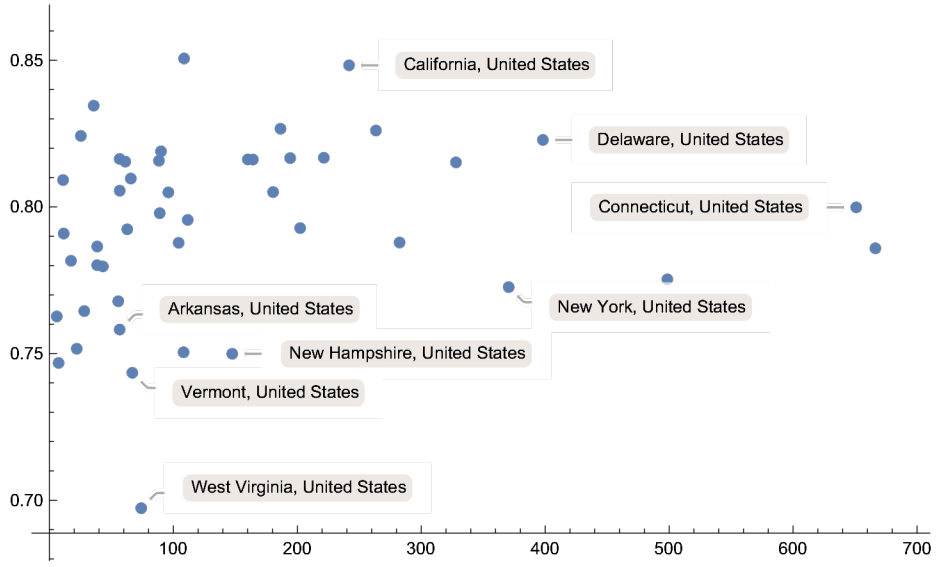
In[3]:=
```
ListPlot[Merge[{EntityValue[Keys[stateStats],
    GDP  nominal | annual | per capita , "EntityAssociation"],
  stateStats[[All, "DistanceEfficiency", "Mean"]]}, Identity]]
```

Out[3]=



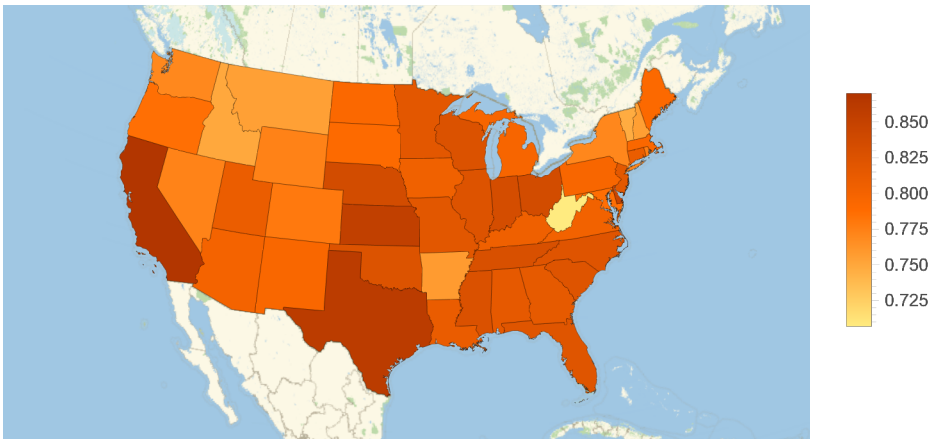
In[4]:=
```
ListPlot[
 Merge[{EntityValue[Keys[stateStats], "PopulationDensity", "EntityAssociation"],
   stateStats[[All, "DistanceEfficiency", "Mean"]]}, Identity]]
```

Out[4]=



In[ ]:=
```
GeoRegionValuePlot[stateStats[[All, "DistanceEfficiency", "Median"]]]
```

Out[ ]=



Country-by-country in Europe

For each country in Europe (excluding Russia), compute the travel efficiency:

```
europeCountryData = generateRandomTables[#, 1000] & /@
    EntityValue[DeleteCases[{Europe GEOGRAPHIC REGION}[{countries}],
      {{Russia COUNTRY}, {Vatican City COUNTRY}}], "Polygon", "EntityAssociation"];
```

In[ ]:=
```
Export[FileNameJoin[{mxDirectory, "europeCountryData.wxf"}],
  europeCountryData, OverwriteTarget → False];
```

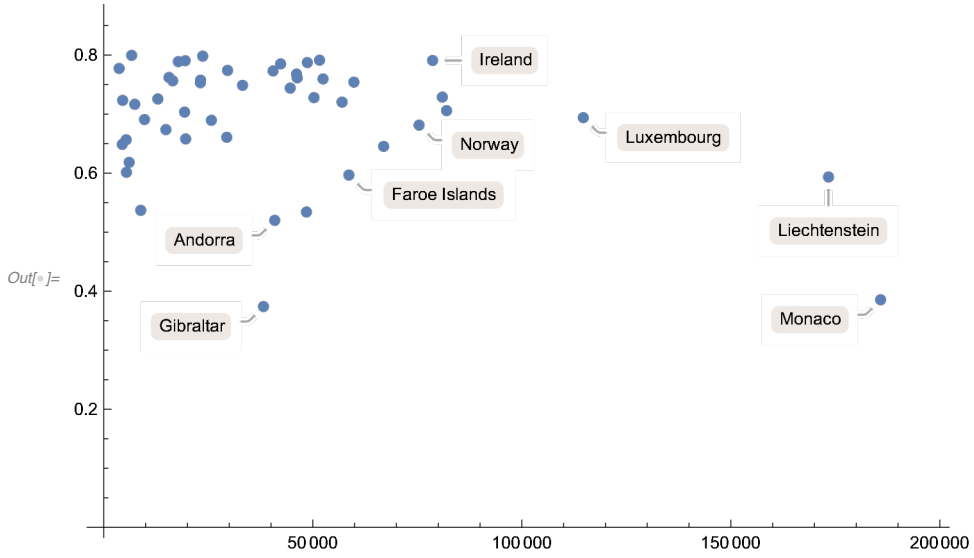
In[32]:=
```
europeCountryData = Import[FileNameJoin[{mxDirectory, "europeCountryData.wxf"}]];
```

```
europeCountryStats = matricesBasicStatistics /@ europeCountryData;
```

In[ ]:=
```
europeCountryStats = ‹|…|› ;
```

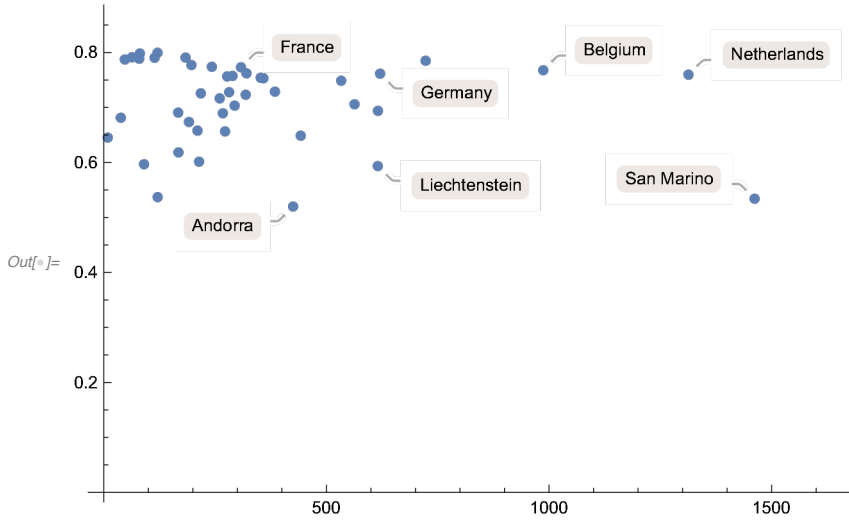
In[ ]:=
```
ListPlot[Merge[{EntityValue[Keys[europeCountryStats], GDP ⊞ nominal | per capita, "EntityAssociation"],
  europeCountryStats[[All, "DistanceEfficiency", "Mean"]]}, Identity]]
```

Out[ ]=



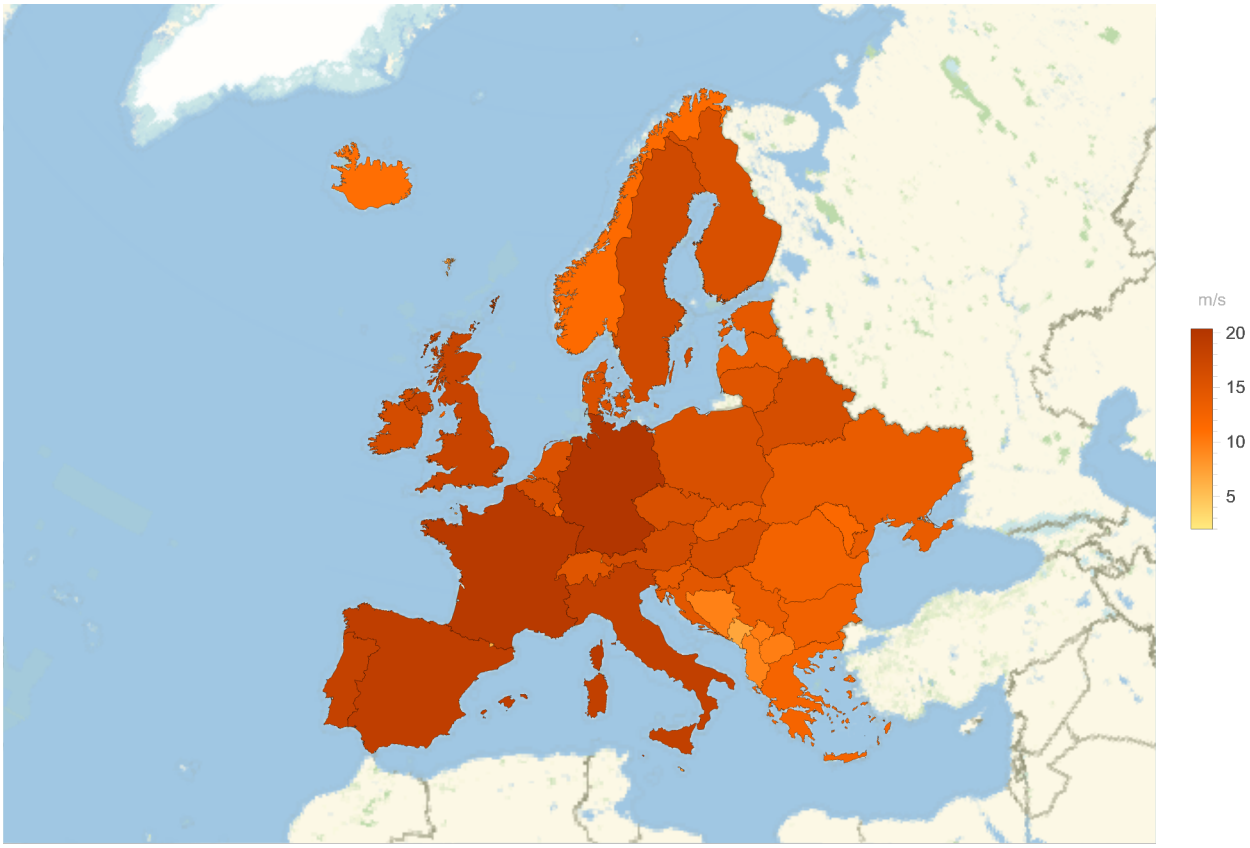
In[ ]:=
```
ListPlot[Merge[
  {EntityValue[Keys[europeCountryStats], population density, "EntityAssociation"],
  europeCountryStats[[All, "DistanceEfficiency", "Mean"]]}, Identity]]
```

Out[ ]=



In[ ]:=
```
GeoRegionValuePlot[europeCountryStats[[All, "TimeEfficiency", "Median"]]]
```

Out[ ]=

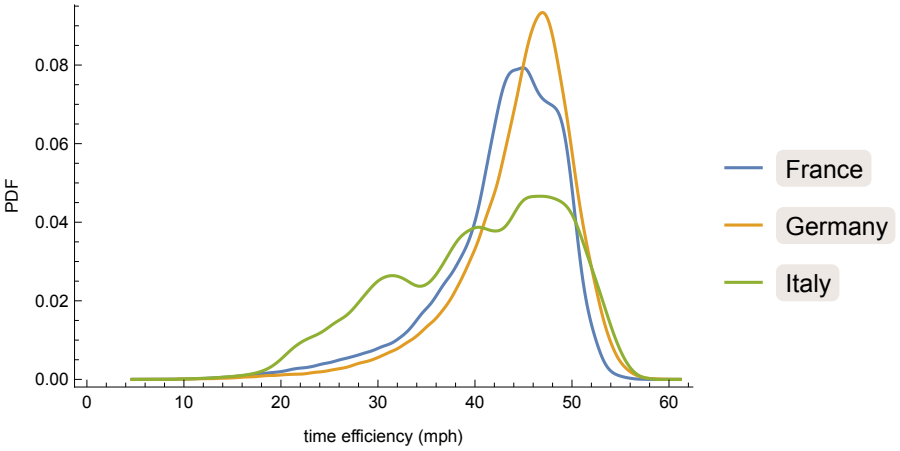


In[ ]:=
```
SmoothHistogram[distanceEfficiency /@ europeCountryData
 (*KeyTake[europeCountryData,{ France COUNTRY , Germany COUNTRY , Italy COUNTRY }]*),
 (*{0.005},*)Automatic, "PDF", Frame → {{True, False}, {True, False}},
 FrameLabel → {"distance efficiency", "PDF"}, PlotLegends → Automatic]
```

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

```
In[ ]:= SmoothHistogram[QuantityMagnitude[timeEfficiency /@ KeyTake[europeCountryData,
          {France COUNTRY, Germany COUNTRY, Italy COUNTRY}]], "Miles" / "Hours"],
        (*{0.005},*)Automatic, "PDF", Frame → {{True, False}, {True, False}},
        FrameLabel → {"time efficiency (mph)", "PDF"}, PlotLegends → Automatic]
```

Out[ ]=



Local sampling

Get the region for the contiguous United States with an orthographic projection:

```
In[ ]:= usRegion = DiscretizeGraphics[
          GeoGridPosition[#, {"Orthographic", "Centering" → United States COUNTRY}] & /@
            Continental US states ADMINISTRATIVE DIVISIONS ["Polygon"]];
```

```
In[49]:= usRegion = MeshRegion[...] + ;
```

For each of 10000 sample points on a triangular lattice in the United States, compute the distance and travel efficiency of a disk with radius 50 miles:

```
In[ ]:= usLocalResults = generateLocalResults[usRegion,
          {"Orthographic", "Centering" → United States COUNTRY}, 10 000];
```

```
In[50]:= usLocalResults = {...} + ;
```

```
usLocalResults = {...} + ;(*Old version*)
```

Smaller radius:

```
In[ ]:= usLocalResults2 = generateLocalResults[usRegion, {"Orthographic",
            "Centering" → United States COUNTRY}, 10 000, Quantity[25, "Miles"]];
```

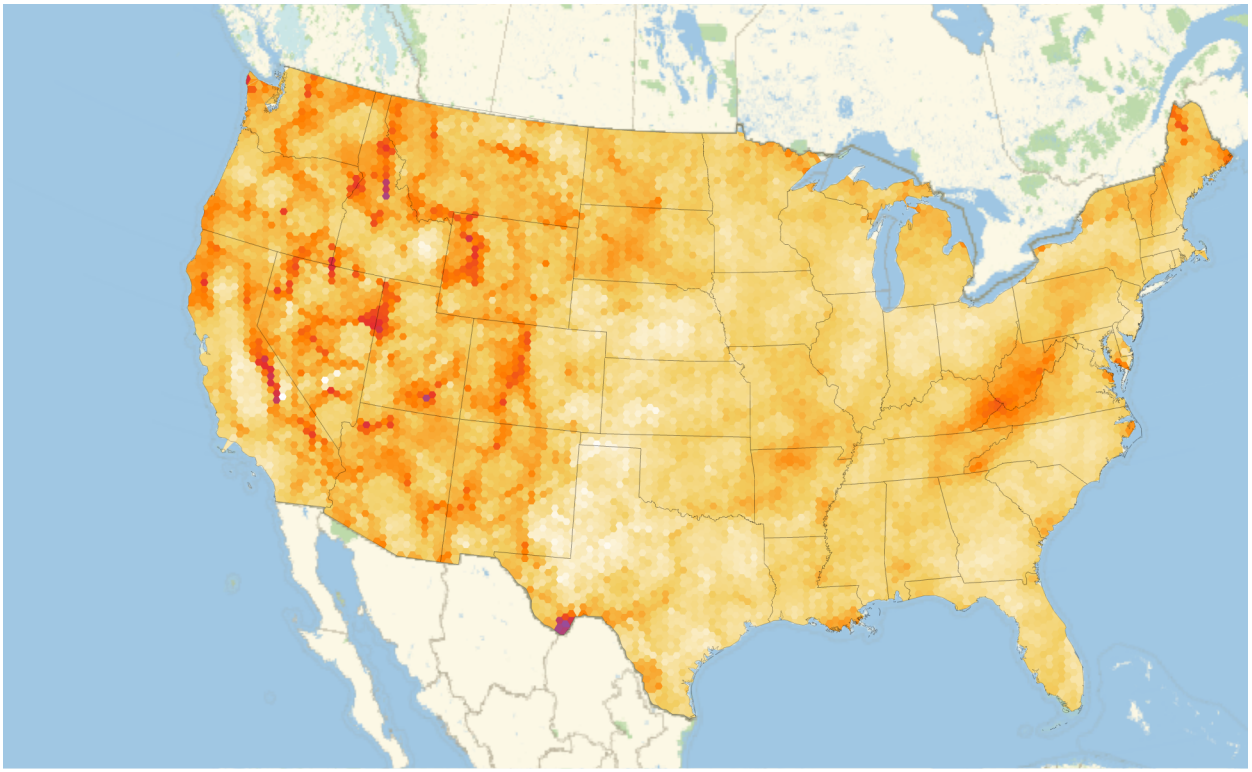
```
In[51]:= usLocalResults2 = {...} + ;
```

In[ ]:=
```
usEquirectangularRegion = RegionUnion[
  BoundaryDiscretizeGraphics[GeoGridPosition[#, "Equirectangular"]] & /@
   Continental US states ADMINISTRATIVE DIVISIONS ["Polygon"]];
```

In[52]:=
```
usEquirectangularRegion = BoundaryMeshRegion[...] ;
```

In[ ]:=
```
Show[{
  GeoDensityPlot[
   #Location → Clip[#DistanceEfficiency["Mean"], {0.4, 1}] & /@ usLocalResults,
   RegionFunction → With[{rmf = RegionMember[usEquirectangularRegion]},
     Function[{x, y, z}, rmf[{y, x}]]], PlotLegends → Automatic,
   InterpolationOrder → 0, OpacityFunction → 1, PlotRange → {0.4, 1},
   ClippingStyle → LightGray, ImageSize → 750],
  GeoGraphics[{EdgeForm[{Opacity[0.25], Thin, Black}], FaceForm[],
    Continental US states ADMINISTRATIVE DIVISIONS ["Polygon"]}]}]
```

Out[ ]=



For Europe:

Download the range of the OSM data:

```
In[ ]:= osmRange = Polygon[ToExpression[StringSplit[StringSplit[
          Import["http://download.geofabrik.de/europe.poly"], "\n"][[3 ;; -3]]]]];
```

```
In[ ]:= osmRange = Polygon[…] + ;
```

Compute the region covered by the OSM data:

```
In[ ]:= europeEquirectangularRegion = RegionIntersection[
     RegionUnion[
      Prepend[
       BoundaryDiscretizeGraphics[GeoGridPosition[#, "Equirectangular"]] & /@
        EntityValue[{Turkey COUNTRY, Cyprus COUNTRY, Georgia COUNTRY}, "Polygon"],
       BoundaryDiscretizeGraphics[GeoGridPosition[Europe GEOGRAPHIC REGION ["Polygon"],
        "Equirectangular"]]]], BoundaryDiscretizeGraphics[osmRange]];
```

```
In[74]:= europeEquirectangularRegion = BoundaryMeshRegion[…] + ;
```

```
In[ ]:= europeRegion = BoundaryMeshRegion[
     GeoGridPosition[GeoGridPosition[MeshCoordinates[europeEquirectangularRegion],
        "Equirectangular"], {"Orthographic", "Centering" → GeoPosition[{52, 13}]}][[
      1]], MeshCells[europeEquirectangularRegion, 1]];
```

```
In[73]:= europeRegion = BoundaryMeshRegion[…] + ;
```

For each of 10000 sample points on a triangular lattice over Europe, compute the distance and travel efficiency of a disk with radius 50 miles:

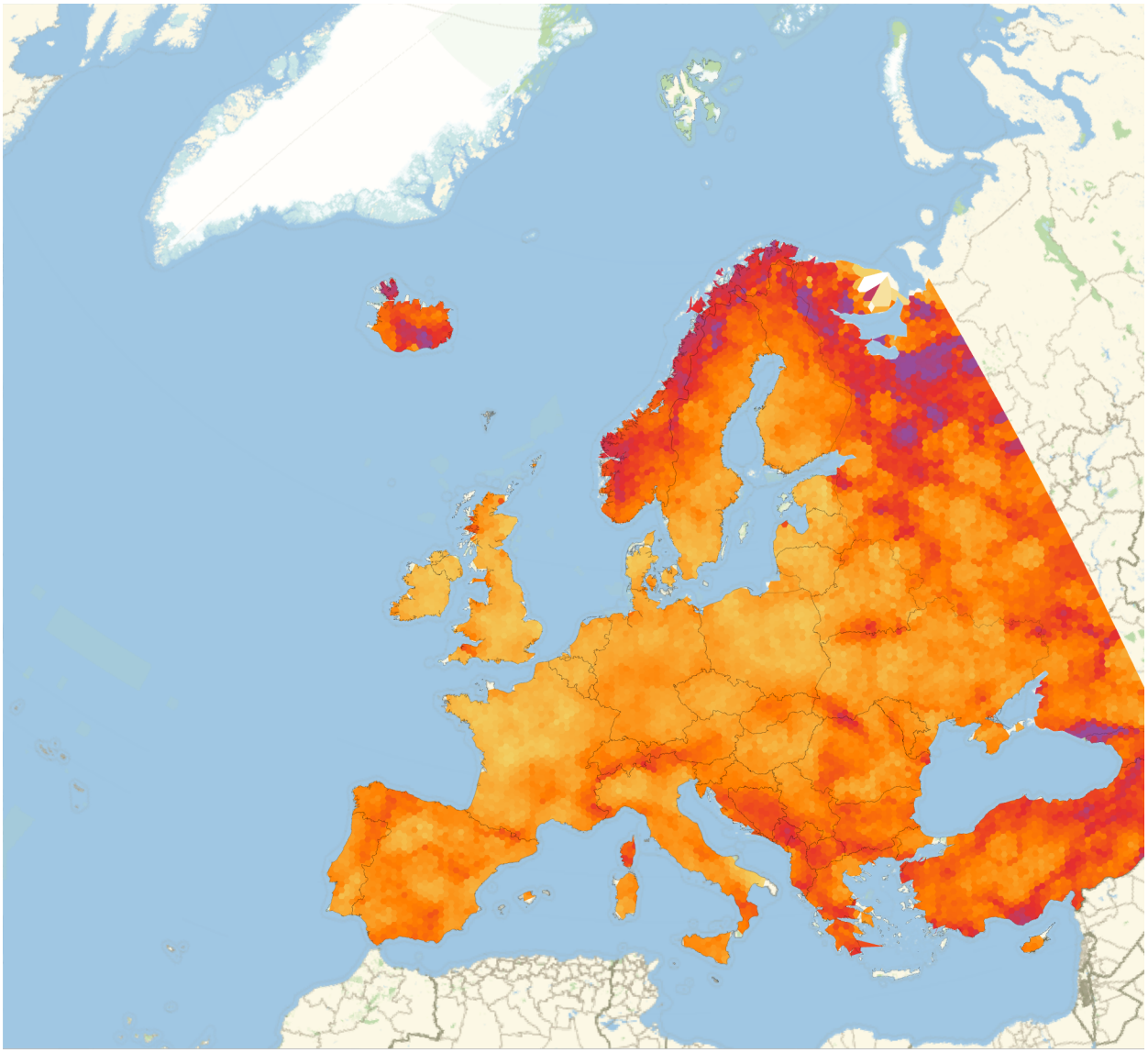
```
In[ ]:= europeLocalResults = generateLocalResults[europeRegion,
     {"Orthographic", "Centering" → GeoPosition[{52, 13}]}, 10 000];
```

```
In[72]:= europeLocalResults = {…} + ;
```

```
europeLocalResults = {…} + ;(*Old version*)
```

In[ ]:=
```
Show[{
 GeoDensityPlot[#Location → Clip[#DistanceEfficiency["Mean"], {0.4, 1}] & /@
   europeLocalResults, PlotLegends → Automatic,
  InterpolationOrder → 0, OpacityFunction → 1, PlotRange → {0.4, 1},
  RegionFunction → With[{rmf = RegionMember[europeEquirectangularRegion]},
    Function[{x, y, z}, rmf[{y, x}]]], ClippingStyle → LightGray, ImageSize → 750],
 GeoGraphics[{EdgeForm[{Opacity[0.25], Thin, Black}], FaceForm[],
   Join[ Europe COUNTRIES ["Polygon"], EntityValue[
    { Turkey COUNTRY , Cyprus COUNTRY , Georgia COUNTRY }, "Polygon"]]}]]
```

Out[ ]=



Experimental

osrmRoute

Unused functions

Bibliography

Ideas