1 Codes

1.1 visualization

```
#!/usr/bin/env python
# coding: utf-8

# ## 加载数据

# In[1]:


import geopandas as gpd
import pandas as pd
food_df_clean = pd.read_csv("../dataset/food_preprocessed.csv")
gdf = gpd.read_file("../map/food.shp")


# ## 相关性系数图

# In[ ]:


import altair as alt
corrs = (
  food_df_clean[["点评数", "口味", "环境", "服务", "人均消费"]]
  .loc[food_df_clean["点评数"] > 5]
  .corr(method="pearson")
  .stack()
  .reset_index()
  .rename(columns={"level_0": "变量1", "level_1": "变量2", 0: "相关系数"})
  .assign(corr_label=lambda x: x['相关系数'].map("{:.2f}".format))
)
corr_chart = (
  alt.Chart(
    corrs
  )
  .mark_rect()
  .encode(
    x=alt.X("变量1:O", title=None),
    y=alt.Y("变量2:O", title=None),
    color="相关系数:Q",
    tooltip=["相关系数"]
  )
  .properties(
    width=300,
    height=300
  )
)

text = (
  corr_chart
  .mark_text()
  .encode(
    text="corr_label",
    color=alt.condition(
      alt.datum.相关系数 > 0.5,
      alt.value("white"),
      alt.value("black")
    )
  )
)

corr_chart + text


# ## 菜品类别平均排行图

# In[18]:


import altair as alt
alt.data_transformers.disable_max_rows()

def generate(field, color_scheme):

  bars = (
    alt.Chart(
      food_df_clean
      .groupby("类别")[field]
      .mean()
      .reset_index()
      .assign(label=lambda x: x[field].map("{:.2f}".format))
    )
    .encode(
      x=alt.X(f"{field}:Q", title=f"平均{field}"),
      y=alt.Y("类别:N", sort="-x"),
      color=alt.Color(f"{field}:Q").scale(scheme=color_scheme).legend(orient="left", title=f"平均{field}"),
      tooltip=[alt.Tooltip(field, title=f"平均{field}")]
    )
    .mark_bar()
    .properties(
      width=400
    )
  )

  text = (
    bars
    .mark_text(
      align="left",
      dx=3
    )
    .encode(
      text="label:Q",
    )
  )

  return bars + text

((generate("人均消费", "reds") & generate("服务", "bluegreen")).resolve_scale(color="independent") | (generate("口味", "plasma") & generate("环境", "spectral")).resolve_scale(color="independent")).resolve_scale(color="independent")


# ## 人均消费热力分布图

# In[96]:


import altair as alt

click_district = alt.selection_point(fields=["行政区"])

choropleth = (
  alt.Chart(
    gdf.rename(columns={"dt_id": "行政区", "dt_name": "行政区名称"})
  )
  .mark_geoshape(
    stroke="white",
    strokeWidth=1
  )
  .encode(
    color=alt.Color("人均消费:Q").scale(scheme="reds").legend(orient="left"),
    tooltip=["行政区名称", "人均消费:Q"],
    opacity=alt.condition(click_district, alt.value(1), alt.value(0.2))
  )
  .transform_lookup(
    lookup="行政区",
    from_=alt.LookupData(data=food_df_clean.groupby("行政区")["人均消费"].mean().reset_index(), key="行政区", fields=["人均消费"])
  )
  .properties(
    height=600,
    width=600,
  )
  .add_params(
    click_district
  )
)

bars = (
  alt.Chart(
    food_df_clean.groupby("行政区")["人均消费"].mean().reset_index()
  )
  .mark_bar()
  .encode(
    x="人均消费",
    opacity=alt.condition(click_district, alt.value(1), alt.value(0.2)),
    color=alt.Color("人均消费:Q").scale(scheme="reds"),
    y=alt.Y("行政区").sort("-x"),
    tooltip=["人均消费:Q"]
  )
  .add_params(
    click_district
  )
  .properties(
    width=500
  )
)

text = (
  bars.mark_text(
    align="left",
    baseline="middle",
    dx=3,
    limit=50
  )
  .encode(
    text="人均消费:Q"
```

```
  )
  .add_params(
    click_district
  )
)
(choropleth | (bars + text)).show()


# In[146]:


import altair as alt

click_district = alt.selection_point(fields=["行政区"], value="A")

choropleth = (
  alt.Chart(
    gdf.rename(columns={"dt_id": "行政区", "dt_name": "行政区名称"})
  )
  .mark_geoshape(
    stroke="white",
    strokeWidth=1,
    fill="red"
  )
  .encode(
    tooltip=["行政区名称"],
    opacity=alt.condition(click_district, alt.value(1), alt.value(0.2))
  )
  .properties(
    height=600,
    width=600,
  )
  .add_params(
    click_district
  )
)
count_bar = (
  alt.Chart(
    pd.concat(
      map(
        lambda x: food_df_clean[food_df_clean["行政区"] == x]["类别"]
          .value_counts()
          .reset_index()
          .assign(行政区=x)
        , food_df_clean["行政区"].unique()
      )
    )
  )
  .mark_bar()
  .transform_filter(
    click_district
  )
  .encode(
    x=alt.X("count:Q", title="总数"),
    # opacity=alt.condition(click_district, alt.value(1), alt.value(0.2)),
    color=alt.Color("count:Q").scale(scheme="reds"),
    y=alt.Y("类别:N").sort("-x"),
    tooltip=[alt.Tooltip("count:Q", title="总数")]
  )
  .add_params(
    click_district
  )
  .properties(
    width=500
  )
)

count_text = (
  count_bar.mark_text(
    align="left",
    baseline="middle",
    dx=3,
    limit=50
  )
  .encode(
    text="count:Q"
  )
  .add_params(
    click_district
  )
)
choropleth | (count_bar + count_text)


# In[20]:


food_df_clean[(food_df_clean["类别"] == "烧烤")]["人均消费"].describe()


# In[50]:


import math
bbq = food_df_clean[(food_df_clean["类别"] == "烧烤")].copy()
bbq = bbq[(bbq["点评数"] > 20) & (bbq["人均消费"] < 500)]
bbq["avg_score"] =  bbq[["口味", "环境", "服务"]].mean(axis=1).apply(math.log2)
bbq = bbq.groupby("avg_score").mean(numeric_only=True).reset_index()
print(bbq)
alt.Chart(bbq).mark_line().encode(
  x="avg_score:Q",
  y="人均消费:Q"
).properties(
  width=9001
).interactive()


# In[117]:


df = food_df_clean.copy()
df = df[df["行政区"] != "L"]
df["点评数_filtered"] = df["点评数"] < 20
df["点评数_filtered"].value_counts()[True]
unpopular_df = df.groupby("行政区").agg(unpopular_rate=("点评数_filtered", lambda x: x.value_counts()[True] / x.count())).reset_index()
print(unpopular_df)
(
  alt.Chart(
    gdf.rename(columns={"dt_id": "行政区", "dt_name": "行政区名称"})
  )
  .mark_geoshape(
    stroke="white",
    strokeWidth=1
  )
  .encode(
    color=alt.Color("unpopular_rate:Q").scale(scheme="reds").legend(orient="left"),
    tooltip=["行政区名称", "unpopular_rate:Q"],
  )
  .transform_lookup(
    lookup="行政区",
    from_=alt.LookupData(data=unpopular_df, key="行政区", fields=["unpopular_rate"])
  ).properties(
    height=600,
    width=600,
  )
)
```

1.2 train

```
#!/usr/bin/env python
# coding: utf-8

# In[17]:


import pandas as pd
food_df = pd.read_csv("../dataset/food_preprocessed.csv")
food_df_clean = food_df[(food_df["人均消费"] <= 1000) & (food_df["点评数"] >= 50)]
food_df_clean


# In[16]:


food_df_clean.size


# In[12]:


from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import json

model_maps = {}

for i in food_df_clean.groupby("行政区"):
  dt_id, dt_df = i
  for j in dt_df.groupby("类别"):
    classification, df = j
    features = food_df_clean[["环境", "口味", "服务", "人均消费"]]
    x, y = features.iloc[:, 0:-1], features.iloc[:, -1]
    model = KMeans(n_clusters=3)
    model.fit(x)
```

```
    model_maps[dt_id + "," + classification]
          = model.cluster_centers_
    print(model.score(x))

print(model_maps)


# In[13]:


for k in model_maps:
  model_maps[k] = list(map(list, model_maps[
      k]))
with open("data.json", mode="w", encoding="
    utf-8") as f:
  f.write(json.dumps(model_maps,
      ensure_ascii=False));


# In[14]:


from sklearn.cluster import KMeans
from sklearn.preprocessing import
    StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
# food_df_clean = food_df_clean[
    food_df_clean["人均消费"] < 300]
# food_df_clean = food_df_clean[
    food_df_clean["点评数"] > 50]
food_df_clean = food_df_clean[food_df_clean[
    "行政区"] == "N"]
food_df_clean = food_df_clean[food_df_clean[
    "类别"] == "浙菜"]

features = food_df_clean[["环境", "口味", "
    服务", "人均消费"]]
x, y = features.iloc[:, 0:-1], features.iloc
    [:, -1]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(
    features)
#pca 降维
a = x.apply(lambda row: row.sum(), axis=1)
pca = PCA(n_components=1)
reduced_features = pca.fit_transform(x)
df = pd.DataFrame({
    "PCA": a,
    "人均消费": y})
# kmeans = KMeans(n_clusters=3, random_state
    =42)
# food_df_clean["Cluster"] = kmeans.
    fit_predict(scaled_features)


plt.rcParams["font.sans-serif"] = ["SimHei"]
sns.scatterplot(x="PCA", y='人均消费', data=
    df, palette="Set2")
plt.show()
```

2 K-means

2.1 k-means

Input: Number of nodes N_A, Number of blocks N_B, Node array A
Output: Seperating result array R
Centers:array[Node], len=N_B;
Randomly initializing Centers;
Cs:Currunt seperation, array[int];
Ls:Last seperation, array[int];
Function `UpdateCs()`:
  Ls ← Cs;
  foreach $i \in [1, N_A]$ **do**
    foreach $j \in [1, N_B]$ **do**
      if $dis(A_i, Cs_i) > dis(A_i, Centers_j)$ **then**
        $Cs_i \leftarrow Centers_j$
      end
    end
  end
return
Function `UpdateCenters()`:
  Csizes:array[int];
  foreach $i \in [1, N_B]$ **do**
    $Csizes_i \leftarrow$ Cs.count(i);
    $Centers_i \leftarrow 0$;
  end
  foreach $i \in [1, N_A]$ **do**
    $Centers_{Cs_i} \leftarrow Centers_{Cs_i} + \frac{A_i}{Csizes_{Cs_i}}$;
  end
return
UpdateCs();
while $Cs \neq Ls$ **do**
  UpdateCenters();
  UppdateCs();
end
return Cs;

Algorithm 1: K-Means