Project Notebook (Python)

Import notebook

Final Project: Voter Analysis for New Hampshire

1.2

states=(AK AL AR AZ CO CT DC DE GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO MS MT NC ND NE NH NJ NM NV OH OK OR RI SC SD TN UT VA VT WA WI WV WY)

for state in "${states[@]}"; do count=$(bq query
--quiet
--format=csv
--use_legacy_sql=false
"SELECT COUNT(*) FROM `pstat-135-235-spring-2025.voterfile.${state}`"
| tail -n +2) if [[-z "$count"]]; then echo "$state : " else echo "$state : ${count//[[:space:]]/}" fi done

The script makes states into a data frame with all state abbreviations and then makes a for loop that goes through each state. It then gets the count for that state abbreviation going through our voter file and using the state abbreviation to grab the count from those states. If there is no table or an error it returns an error code, if everything is fine it returns the count.

1.3

The notebook is running on the BigQuery compute engine in the US Region. The open job page revealed execution detailsm such as byte processed, whether or not results were cached and time taken.

;

Using the Cloud Shell was very efficeint for querying all 50 states as the script was ideal for automating repetitive tasks. The downside of the script was its lack of visualization as well as being hard to explore and debug.

Using the BigQuery Notebook allowed for integration of both SQL and Python, as well as data visualization. It was extremely easy to explore, transform and plot the data, but the drawback was that it depended on runtime availability as well as a longer setup.

Overall, Cloud Shell was better for automation while BigQuery Notebooks was easier for exploration and visualization.

```
import pyspark.pandas as ps
import pandas as pd
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder
    .appName("PandasOnSpark Example")
    .getOrCreate()
)

df = spark.read.table("New_Hampshire")
display(df)
```

▸ df: pyspark.sql.dataframe.DataFrame = [SEQUENCE: long, LALVOTERID: string ... 439 more fields]

Table

Section 2

Different Pyspark Panda APIs for data exploration

```
psdf = df.pandas_api()

null_fraction = psdf.isnull().sum() / len(psdf)

mostly_null_cols = null_fraction[null_fraction > 0.75].index.tolist()

psdf_cleaned = psdf.drop(columns=mostly_null_cols)

df_cleaned = psdf_cleaned.to_spark()
```

▸ df_cleaned: pyspark.sql.dataframe.DataFrame = [SEQUENCE: long, LALVOTERID: string ... 262 more fields]

1). .pandas_api() allowed for the conversion of a PySpark Dataframe into a Pandas-on-Spark DataFrame, which allows for Panda-like methods to be used on the DataFrame.

2). .isnull() returns a Boolean mask whihc indicates where values are null, which is useful in seeing the column and row of null values.

3). .sum() sums up the entries of whatever is needed. When combined with .isnull(), it counts the amount of null values per column.

4). len() gives the total number of rows in the Pandas-on-Spark DataFrame, when combined with .isnull() and .sum() it computed the null fraction per column.

5). .drop() drops the identified columns from Pandas-on-Spark DataFrame. It is useful for dropping null columns from a DataFrame.

```
psdf = df.pandas_api()

primary_is_y = psdf["Primary_2008"] == "Y"
general_is_y = psdf["General_2012"] == "Y"

conf_matrix = (
        psdf.assign(primary_y=primary_is_y, general_y=general_is_y)
        .groupby(["primary_y", "general_y"])
        .size()
        .unstack()
        .fillna(0)
        .rename(index={True: "primary_Y", False: "primary_not_Y"},
                columns={True: "general_Y", False: "general_not_Y"})
)

print(conf_matrix)
```

```
general_y       false    true
primary_y
primary_Y        3826   77514
primary_not_Y  416864  484792
```

Voter Turnout by Age Groups

```
def age_to_group(age):
    if age < 30:
        return '20s'
    elif age < 40:
        return '30s'
    elif age < 50:
        return '40s'
    elif age < 60:
        return '50s'
    elif age < 70:
        return '60s'
    elif age < 80:
        return '70s'
    else:
        return '80+'

psdf['age_group'] = psdf['Voters_Age'].apply(age_to_group)
```

```
import matplotlib.pyplot as plt
import plotly.express as px

primary_cols = [col for col in df.columns if col.startswith("PRI") and "GEN"
not in col]
primary_cols.append("age_group")

nh_subset = psdf[primary_cols]
nh_long = nh_subset.melt(
    id_vars='age_group',
    var_name='year',
    value_name='vote_value'
)

nh_long['year'] = nh_long['year'].apply(lambda x:
int(''.join(filter(str.isdigit, x))[:4]))

counts_by_party = (
    nh_long
      .dropna(subset=["vote_value"])                    # keep only real
ballots: O, R, D
      .groupby(["year", "age_group", "vote_value"])
      .size()
      .reset_index(name="count")                         # DataFrame: year |
age_group | vote_value | count
)

counts_pd = counts_by_party.to_pandas()
pivot_pd = (
    counts_pd
      .pivot_table(
          index=["year", "age_group"],
          columns="vote_value",
          values="count",
          fill_value=0
      )
      .reset_index()
)

pivot_pd["total_voters"] = pivot_pd[["O", "R", "D"]].sum(axis=1)

plot_df = pivot_pd[["year", "age_group", "total_voters"]].rename(
    columns={"age_group": "Age Group", "total_voters": "Voter Count"}
)
```

```
fig = px.line(
    plot_df,
    x="year",
    y="Voter Count",
    color="Age Group",
    markers=True,
    title="Primary Election Turnout by Age Group in New Hampshire"
)
fig.update_layout(
    xaxis_title="Year",
    yaxis_title="Total Primary Voters",
    template="plotly_white"
)
fig.show()
```

The line chart "Voter Turnout by Age Group (Primary Elections) in New Hampshire" visualizes the voter turnout for the primaries in New Hampshire from 2000-2020. The x-axis represents the years that elections occur, which is every even year, while the y axis shows the number of voters. Each line represents individuals in different age intervals, starting with the 20s and ending with people 80 and older.

The visualization reveals that in 2020, there was a huge spike for voter turn out, with individuals in their 60s having the biggest turn out, at about 74.47K. During the same year, the smallest turnout were individuals in their 20s at barely 7961 voters. This suggests that younger voters tend to vote less. However this year is also the year with the highest voter turnout in general. This could have been because of the Covid-19 pandemic and more people wanting their vote and voices heard during the pandemic. Interestingly, in 2004, there were barely any votes from any age group, with all of them being extremely close to zero. This is the only year to have all age groups with similar turnouts.

```
    psdf['PRI_BLT_2004'].value_counts().to_pandas()
```

```
/databricks/spark/python/pyspark/pandas/base.py:1437: FutureWarning:

The resulting Series will have a fixed name of 'count' from 4.0.0.


O    2644
D     950
R     701
Name: PRI_BLT_2004, dtype: int64
```

```
from urllib.request import urlopen
import json
import pandas as pd
import plotly.express as px

psdf_nh = psdf[["County", "General_2016"]]

psdf_nh = psdf_nh.assign(
    County = psdf_nh["County"].str.upper().str.strip()
)

psdf_nh = psdf_nh.assign(
    voted_2016 = psdf_nh["General_2016"].map({ "Y": 1, "N": 0 })
)

psdf_grouped = (
    psdf_nh
    .groupby("County", as_index=False)
    .agg({ "voted_2016": "mean" })
)

psdf_grouped = psdf_grouped.rename(
    columns={ "County": "county_name", "voted_2016": "turnout_frac" }
)

psdf_grouped = psdf_grouped.assign(turnout_rate = psdf_grouped["turnout_frac"]
* 100)

nh_fips = {
    "BELKNAP":       "33001",
    "CARROLL":       "33003",
    "CHESHIRE":      "33005",
    "COOS":          "33007",
    "GRAFTON":       "33009",
    "HILLSBOROUGH":  "33011",
    "MERRIMACK":     "33013",
    "ROCKINGHAM":    "33015",
    "STRAFFORD":     "33017",
    "SULLIVAN":      "33019"
}
psdf_grouped = psdf_grouped.assign(
    fips = psdf_grouped["county_name"].map(nh_fips)
)
```

```
psdf_grouped = psdf_grouped.dropna(subset=["fips"])

pdf_county = psdf_grouped.to_pandas()

with urlopen("https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counties-fips.json") as r:
    geojson_all = json.load(r)

nh_geojson = {
    "type": "FeatureCollection",
    "features": [
        feature for feature in geojson_all["features"]
        if feature["id"].startswith("33")
    ]
}


fig = px.choropleth(
    pdf_county,
    geojson=nh_geojson,
    locations="fips",            # matches our "fips" column
    color="turnout_rate",        # % of registered voters who voted in 2016
    color_continuous_scale="Viridis",
    range_color=(0, 100),
    scope="usa",
    labels={ "turnout_rate": "Turnout % (2016)" },
    title="NH 2016 General Election Turnout by County"
)

fig.update_traces(marker_line_width=1, marker_line_color="black")

fig.update_geos(
    visible=False,
    projection_type="mercator",
    projection_scale=5.5,
    center={"lat": 43, "lon": -71.5},
    bgcolor="lightgray"
)

fig.update_layout(
    height=600,
    width=700,
    margin={"r":0, "t":50, "l":0, "b":0},
    plot_bgcolor="lightgray"
)
```

```
    fig.show()

    pdf_county = psdf_grouped.to_pandas()

    print("pdf_county.head():")
    print(pdf_county.head())
    print("pdf_county.dtypes:")
    print(pdf_county.dtypes)
    print("How many rows? →", len(pdf_county))

    with urlopen("https://raw.githubusercontent.com/plotly/datasets/master/geojson-
    counties-fips.json") as r:
        geojson_all = json.load(r)

    nh_geojson = {
        "type": "FeatureCollection",
        "features": [
            feature for feature in geojson_all["features"]
            if feature["id"].startswith("33")
        ]
    }

    print("Number of NH polygons in geojson:", len(nh_geojson["features"]))
```

```
pdf_county.head():
  county_name  turnout_frac  turnout_rate   fips
0        COOS           1.0         100.0  33007
1     CARROLL           1.0         100.0  33003
2    CHESHIRE           1.0         100.0  33005
3  ROCKINGHAM           1.0         100.0  33015
4     GRAFTON           1.0         100.0  33009
pdf_county.dtypes:
county_name      object
turnout_frac    float64
turnout_rate    float64
fips             object
dtype: object
How many rows? → 10
Number of NH polygons in geojson: 10
```

```
from urllib.request import urlopen
import json
import requests
import pandas as pd
import plotly.express as px

nh = psdf.copy()
nh = nh[nh["PRI_BLT_2020"].isin(["R", "D", "O"])]
nh["County"] = nh["County"].str.upper().str.strip()

turnout = nh.groupby("County").size().reset_index(name="total_ballots")

county_fips_map = {
    "BELKNAP":       "33001",
    "CARROLL":       "33003",
    "CHESHIRE":      "33005",
    "COOS":          "33007",
    "GRAFTON":       "33009",
    "HILLSBOROUGH":  "33011",
    "MERRIMACK":     "33013",
    "ROCKINGHAM":    "33015",
    "STRAFFORD":     "33017",
    "SULLIVAN":      "33019"
}
turnout["fips"] = turnout["County"].map(county_fips_map)
turnout = turnout.dropna(subset=["fips"])
turnout_pdf = turnout.to_pandas()

url = "https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counties-fips.json"
geojson_all = requests.get(url).json()
nh_geojson = {
    "type": "FeatureCollection",
    "features": [
        feature
        for feature in geojson_all["features"]
        if feature["id"].startswith("33")
    ]
}

fig = px.choropleth(
    turnout_pdf,
    geojson=nh_geojson,
    locations="fips",
```

```
        color="total_ballots",
        color_continuous_scale="purp",
        labels={"total_ballots": "Ballots"},
        title="2020 New Hampshire Primary Voter Turnout by County"
    )

    fig.update_traces(marker_line_width=1, marker_line_color="black")

    fig.update_geos(
        fitbounds="locations",
        visible=False
    )

    fig.update_layout(
        height=650,
        width=900,
        margin={"r": 0, "t": 50, "l": 0, "b": 0},
        plot_bgcolor="lightgray"
    )

    fig.show()
```

The choropleth map visualizes the 2020 New Hampshire Primary voter turnout bu county, summing up Democratic and Republican ballots. The counties that are shaded darker purple represent counties that have higher voter turnouts, while counties with a lighter shade had lower voter turnouts. Through the visualization, it is apparent that Hillsborough and Rockingham have the highest voter turnout, while Coos and Sullivan had the lowest voter turnout.

```
party_counts = nh.groupby(["County",
"PRI_BLT_2020"]).size().reset_index(name='Counts').pivot(index='County',
columns='PRI_BLT_2020', values='Counts').fillna(0)
party_counts["Total"] = party_counts["R"] + party_counts["D"]
party_counts["Pct_Rep"] = party_counts["R"] / party_counts["Total"] * 100
party_counts.reset_index(inplace=True)
party_counts["fips"] = party_counts["County"].map(county_fips_map)
party_counts = party_counts.dropna(subset=["fips"])
party_counts_pdf = party_counts.to_pandas()

fig = px.choropleth(
    party_counts_pdf,
    geojson=nh_geojson,
    locations="fips",
    color="Pct_Rep",
    color_continuous_scale="reds",
    labels={"Pct_Rep": "% Republican"},
    title="2020 New Hampshire Primary: % Republican Voters by County"
)
fig.update_traces(marker_line_width=1, marker_line_color='black')
fig.update_geos(
    fitbounds="locations",
    visible=False
)
fig.update_layout(
    height=650,
    width=900,
    margin={"r":0, "t":50, "l":0, "b":0},
    plot_bgcolor="lightgray"
)

fig.show()
```

```
party_counts = nh.groupby(["County",
"PRI_BLT_2020"]).size().reset_index(name='Counts').pivot(index='County',
columns='PRI_BLT_2020', values='Counts').fillna(0)
party_counts["Total"] = party_counts["R"] + party_counts["D"]
party_counts["Pct_Dem"] = party_counts["D"] / party_counts["Total"] * 100
party_counts.reset_index(inplace=True)
party_counts["fips"] = party_counts["County"].map(county_fips_map)
party_counts = party_counts.dropna(subset=["fips"])
party_counts_pdf = party_counts.to_pandas()

fig = px.choropleth(
    party_counts_pdf,
    geojson=nh_geojson,
    locations="fips",
    color="Pct_Dem",
    color_continuous_scale="Blues",
    labels={"Pct_Dem": "% Democrat"},
    title="2020 New Hampshire Primary: % Democrat Voters by County"
)
fig.update_traces(marker_line_width=1, marker_line_color='black')
fig.update_geos(
    fitbounds="locations",
    visible=False
)
fig.update_layout(
    height=650,
    width=900,
    margin={"r":0, "t":50, "l":0, "b":0},
    plot_bgcolor="lightgray"
)

fig.show()
```

```
vote_counts = nh.groupby(["County",
"PRI_BLT_2020"]).size().reset_index(name='Counts').pivot(index='County',
columns='PRI_BLT_2020', values='Counts').fillna(0)
vote_counts["Total"] = vote_counts["R"] + vote_counts["D"] + vote_counts["O"]
vote_counts["Lean"] = (vote_counts["R"] - vote_counts["D"]) /
vote_counts["Total"] * 100  # +R / -D
vote_counts.reset_index(inplace=True)
vote_counts["fips"] = vote_counts["County"].map(county_fips_map)
vote_counts = vote_counts.dropna(subset=["fips"])
vote_counts_psdf = vote_counts.to_pandas()

fig = px.choropleth(
    vote_counts_psdf,
    geojson=nh_geojson,
    locations="fips",
    color="Lean",
    color_continuous_scale="RdBU",
    range_color = [-100, 100],
    labels={"Lean": "Partisan Lean"},
    title="2020 New Hampshire Primary: Partisan Lean by County"
)
fig.update_traces(marker_line_width=1, marker_line_color='black')
fig.update_geos(
    fitbounds="locations",
    visible=False
)
fig.update_layout(
    height=650,
    width=900,
    margin={"r":0, "t":50, "l":0, "b":0},
    plot_bgcolor="lightgray"
)

fig.show()
```

```
    vote_counts = nh.groupby(["County",
    "PRI_BLT_2020"]).size().reset_index(name='Counts').pivot(index='County',
    columns='PRI_BLT_2020', values='Counts').fillna(0)
    party_counts.columns
```

```
Index(['County', 'D', 'O', 'R', 'Total', 'Pct_Dem', 'fips'], dtype='object', name='PRI_
BLT_2020')
```

The two above choropleth maps visualize how each county in New Hampshire voted in the 2020 Primary election, that being either Democratic, Republican or Non Partisan. It can be seen that in the county of Belknap, there was almost 61 percent of Republican votes. In the counties of Grafton and Cheshire, there was a strong number of Democrats that voted with both counties having about 61%. Non-Partisan votes are normally lower compared to both other parties, and this can be seen in the visualization as the counties of Belknap, Grafton and Cheshire had between 21 and 22 percent, which also coincides with the counties that were the highest leaning for either Democrats or Republicans.

Section 3: Supervised and Unsupervised Learning

QUESTION #3: Supervised and Unsupervised Learning

Supervised Learning Model (1. Logistic Regression) income level during gen. elections

```
from pyspark.ml.classification import LogisticRegression
from pyspark.sql.functions import col, when
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator


#new column for voters during presidential election

df_cleaned = df_cleaned.withColumn(
    'voted_in_presidential',
    when(
        (col('General_2008') == 'Y') |
        (col('General_2012') == 'Y') |
        (col('General_2016') == 'Y') |
        (col('General_2010') == 'Y'),
        1
    ).otherwise(0)
)
psdf_cleaned = df_cleaned.pandas_api()

#relevant columns
psdf_cleaned['age_group'] = psdf_cleaned['Voters_Age'].apply(age_to_group)
df_cleaned = psdf_cleaned.to_spark()
age_indexer = StringIndexer(
    inputCol='age_group',
    outputCol='age_group_index',
    handleInvalid='keep'
)

income_indexer = StringIndexer(
    inputCol='Household_Income',
    outputCol='income_index',
    handleInvalid='keep'
)

assembler = VectorAssembler(
    inputCols=['age_group_index', 'income_index'],
    outputCol='features'
)

lr = LogisticRegression(featuresCol='features',
labelCol='voted_in_presidential')
```

```
pipeline = Pipeline(stages=[age_indexer, income_indexer, assembler, lr])

#training and testing
train_data, test_data = df_ml.randomSplit([0.7, 0.3], seed=42)

#logistic regression
lr = LogisticRegression(featuresCol='features',
labelCol='voted_in_presidential')

pipeline = Pipeline(stages=[age_indexer, income_indexer, assembler, lr])

#train model
model = pipeline.fit(train_data)

#predict on test
predictions = model.transform(test_data)

#check model
evaluator = BinaryClassificationEvaluator(
    labelCol='voted_in_presidential',
    rawPredictionCol='rawPrediction',
    metricName='areaUnderROC'
)

auc = evaluator.evaluate(predictions)
print(f"Test AUC: {auc:.3f}")

#interpret coefficients
lr_model = model.stages[-1]
print(f"Coefficients: {lr_model.coefficients}")
print(f"Intercept: {lr_model.intercept}")
```

▸ df_cleaned: pyspark.sql.dataframe.DataFrame = [SEQUENCE: long, LALVOTERID: string ... 264 more fields]

▸ predictions: pyspark.sql.dataframe.DataFrame

▸ test_data: pyspark.sql.dataframe.DataFrame = [voted_in_presidential: integer, age_group: string ... 1 more field]

▸ train_data: pyspark.sql.dataframe.DataFrame = [voted_in_presidential: integer, age_group: string ... 1 more field]

🏃 View run luxuriant-shark-950 at: https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172208717/runs/9f60232c624a4625bee36ef3663ee0fd (https://116865565867

```
0067.7.gcp.databricks.com/ml/experiments/2316398172208717/runs/9f60232c624a4625bee36ef3
663ee0fd)
🧪 View experiment at: https://1168655658670067.7.gcp.databricks.com/ml/experiments/23
16398172208717 (https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172
208717)
Test AUC: 0.713
Coefficients: [0.3673602838556813,-0.0005578329505072359]
Intercept: 0.8027028717097339
```

Based on results:

Election campaigns target groups that are most likely to vote, so identifying high turnout groups is important to any candidate's campaign. Once identified they can choose to allocate more time and resources to the high participant groups or begin outreach efforts to target lower turnout groups. Generally speaking higher turnout groups have the resources to get to poll booths during elections without socioeconomic interference. Some initiatives for lowering voter turnout include expanding voter registration, transportation assistance, and promoting voter's right education and election information.

```
Supervised Learning Model (Logistic Regression)
Gender impact on voter turnout
```

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import col, when

# Create label column
df_cleaned = df_cleaned.withColumn(
    'voted_in_presidential',
    when(
        (col('General_2008') == 'Y') |
        (col('General_2012') == 'Y') |
        (col('General_2016') == 'Y') |
        (col('General_2010') == 'Y'),
        1
    ).otherwise(0)
)

# Select gender and target
df_gender = df_cleaned.select(
    'voted_in_presidential',
    'Voters_Gender'
).na.drop()

# Corrected inputCol name
gender_indexer = StringIndexer(inputCol='Voters_Gender',
outputCol='gender_index', handleInvalid='keep')

assembler = VectorAssembler(
    inputCols=['gender_index'],
    outputCol='features'
)

# Train/test split
train_data, test_data = df_gender.randomSplit([0.7, 0.3], seed=42)

# Define and fit pipeline
lr = LogisticRegression(featuresCol='features',
labelCol='voted_in_presidential')
pipeline = Pipeline(stages=[gender_indexer, assembler, lr])

model = pipeline.fit(train_data)
predictions = model.transform(test_data)
```

```
    # Evaluate
    evaluator = BinaryClassificationEvaluator(
        labelCol='voted_in_presidential',
        rawPredictionCol='rawPrediction',
        metricName='areaUnderROC'
    )

    auc = evaluator.evaluate(predictions)
    print(f"Test AUC: {auc:.3f}")

    # Model interpretation
    lr_model = model.stages[-1]
    print(f"Coefficients: {lr_model.coefficients}")
    print(f"Intercept: {lr_model.intercept}")
```

▸ df_cleaned: pyspark.sql.dataframe.DataFrame = [SEQUENCE: long, LALVOTERID: string ... 264 more fields]

▸ df_gender: pyspark.sql.dataframe.DataFrame = [voted_in_presidential: integer, Voters_Gender: string]

▸ predictions: pyspark.sql.dataframe.DataFrame

▸ test_data: pyspark.sql.dataframe.DataFrame = [voted_in_presidential: integer, Voters_Gender: string]

▸ train_data: pyspark.sql.dataframe.DataFrame = [voted_in_presidential: integer, Voters_Gender: string]

```
🏃 View run intrigued-tern-609 at: https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172208717/runs/50666aa94b264828ac39f609223bb138 (https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172208717/runs/50666aa94b264828ac39f609223bb138)
🧪 View experiment at: https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172208717 (https://1168655658670067.7.gcp.databricks.com/ml/experiments/2316398172208717)
Test AUC: 0.516
Coefficients: [-0.11601035642880755]
Intercept: 1.3511963462183894
```

Based on results:

Campaign strategies often overlook subtle gender-based turnout trends. If one gender is found to consistently vote at higher rates, this insight can be used to tailor outreach messages more effectively by targeting that gender's interests, mobilize

underrepresented gender groups through targeted media, and adapt canvassing strategies for regions with skewed gender voter turnout. High AUC values indicate the model's success in identifying patterns in gender-based voting behavior. In a campaign context, data-driven tools like this can drive resource optimization and improve civic engagement for marginalized demographics.