Coding_output

April 28, 2025

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
```

[ ]:
```
df = pd.read_csv('pokemon_data.csv')
```

[ ]:
```
df_c = pd.DataFrame(
    # Removing duplicate rows
    df.drop_duplicates()
    # removing empty rows
    .dropna(how='all')
    # Removing specific columns
    .drop(columns=[])
    .reset_index(drop=True)
)

# Renaming column titles: replacing _ with spaces and capitalising each word
df_c.columns = df_c.columns.str.replace('_', ' ').str.title()
# Capitalising 'Id' and 'Hp' column title, saving changes
df_c.rename(columns={'Id':'ID', 'Hp':'HP', 'Sp. Attack': 'Special Attack', 'Sp.␣
↪Defense':'Special Defense'}, inplace=True)

# Adding Total Weakness Column
list_weaknesses = ['Normal Weakness', 'Fire Weakness', 'Water Weakness',␣
↪'Electric Weakness',
    'Grass Weakness', 'Ice Weakness', 'Fighting Weakness',  'Poison Weakness',
    'Ground Weakness', 'Flying Weakness', 'Psychic Weakness', 'Bug Weakness',
    'Rock Weakness',   'Ghost Weakness', 'Dragon Weakness', 'Dark Weakness',
    'Steel Weakness', 'Fairy Weakness']

df_c['Total Weakness'] = df_c[list_weaknesses].sum(axis=1)
```

[ ]:
```
# Making dictionary to store dfs
type_df_dict = {}

# Go through each unique value in the Type 1 column
```

```
for type_value in df_c['Type 1'].unique():
    # Filter the DataFrame for the rows same as current type
    type_df = df_c[df_c['Type 1'] == type_value]

    # Store the DataFrame in a dictionary with the type as the key
    type_df_dict[f"df_type_{type_value}"] = type_df


type_df_names = list(type_df_dict.keys())
print(type_df_names)
```

```
['df_type_Grass', 'df_type_Fire', 'df_type_Water', 'df_type_Bug',
'df_type_Normal', 'df_type_Poison', 'df_type_Electric', 'df_type_Ground',
'df_type_Fairy', 'df_type_Fighting', 'df_type_Psychic', 'df_type_Rock',
'df_type_Ghost', 'df_type_Ice', 'df_type_Dragon', 'df_type_Dark',
'df_type_Steel', 'df_type_Flying']
```

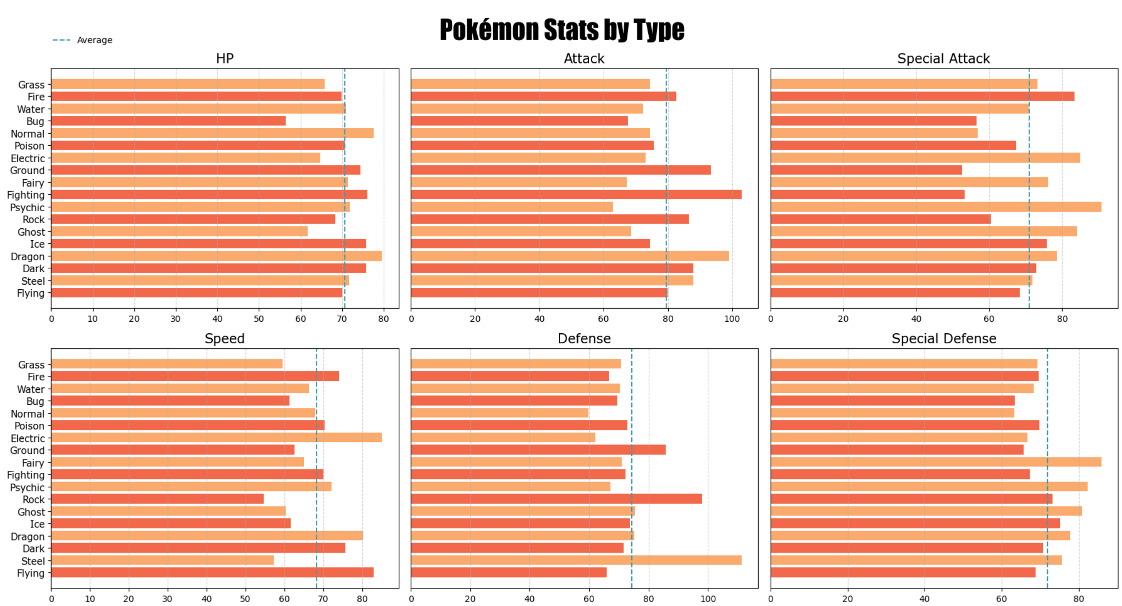
[ ]:
```
all_stats_base = ['Base Stats', 'HP', 'Attack', 'Special Attack', 'Speed',␣
 ↪'Defense', 'Special Defense']

type_df_mean_dict = {}

# Go through each unique value in the Type 1 column
for type_value in df_c['Type 1'].unique():
    # Filter the DataFrame for the rows same as current type
    type_df = df_c[df_c['Type 1'] == type_value]

    mean_values = type_df[all_stats_base].mean().round()

    # Create a new dataframe with the mean values
    mean_df = pd.DataFrame(mean_values).transpose()  # Convert the Series to a␣
 ↪DataFrame
    mean_df['Type 1'] = type_value  # Add the type as a column

    # Store the new mean dataframe in the dictionary
    type_df_mean_dict[f"df_type_{type_value}_mean"] = mean_df

type_df_mean_names = list(type_df_mean_dict.keys())
print(type_df_mean_names)
```

```
['df_type_Grass_mean', 'df_type_Fire_mean', 'df_type_Water_mean',
'df_type_Bug_mean', 'df_type_Normal_mean', 'df_type_Poison_mean',
'df_type_Electric_mean', 'df_type_Ground_mean', 'df_type_Fairy_mean',
'df_type_Fighting_mean', 'df_type_Psychic_mean', 'df_type_Rock_mean',
'df_type_Ghost_mean', 'df_type_Ice_mean', 'df_type_Dragon_mean',
'df_type_Dark_mean', 'df_type_Steel_mean', 'df_type_Flying_mean']
```

[ ]:
```python
# Creating new dataframe using only columns needed
df_stats = df_c[['HP', 'Attack', 'Special Attack', 'Defense', 'Special␣
↪Defense', 'Speed', 'Base Stats', 'Type 1']]

type_order = df_stats['Type 1'].unique()

# Group by 'Type 1' and calculate mean of all stats
plot_df_stats = df_stats.groupby('Type 1').mean(numeric_only=True).reset_index()

# Order by custom type order
plot_df_ordered = plot_df_stats.set_index('Type 1').loc[type_order].
↪reset_index()

# Stats setup
all_stats = ['HP', 'Attack', 'Special Attack', 'Speed', 'Defense', 'Special␣
↪Defense']

overall_averages = plot_df_stats[all_stats].mean()
```

[ ]:
```python
# Plotting parallel bar chart to compare stats across types of pokemon

# importing Line2D to create line for legend
from matplotlib.lines import Line2D

# setting custom fonts
impact_font = {'fontname':'Impact'}
verdana_font = {'fontname': 'Verdana'}

# Plot figure
fig = plt.figure(figsize=(18, 18))
# Adding title: chnaging font, size, and position
fig.suptitle('Pokémon Stats by Type', **impact_font, size=30, y=0.76)

# Create custom line for the legend
avg_line = Line2D([0], [0], color='#419EAE', linestyle='--', linewidth=1.5,␣
↪label='Average')

# Add legend to plot
fig.legend(handles=[avg_line], loc='upper right', fontsize=10,␣
↪bbox_to_anchor=(0.11, 0.75), frameon=False)

# Plotting graphs for individual stats
# Creating loop to plot all charts
for i, stat in enumerate(all_stats):
    row = 1 + i // 3
    col = i % 3
    ax = plt.subplot2grid((4, 3), (row, col))
```

```
    values = plot_df_ordered.set_index('Type 1')[stat]

    colours = ['#FAAA6D', '#F2684A']
    bar_colours = [colours[i % 2] for i in range(len(values))]

    # Customise charts
    ax.barh(type_order, values, color=bar_colours)
    ax.set_title(f'{stat}', fontsize=15, **verdana_font)
    ax.invert_yaxis()
    ax.grid(axis='x', linestyle='--', alpha=0.6)
    ax.axvline(overall_averages[stat], color='#419EAE', linestyle='--', linewidth=1.5)

    # For the first column in each row, show y-ticks with the Pokémon types
    if col == 0:
        ax.set_yticks(range(len(type_order)))
        ax.set_yticklabels(type_order, fontsize=11, **verdana_font)

    # For other columns, hide the y-ticks (type)
    else:
        ax.set_yticks([])

# Printing plot
plt.tight_layout()
plt.show()
```



[ ]:
```
# creating summary stats table and rounding
summary_stats = df_c[all_stats_base].describe().round()
summary_stats
```

[ ]:
```
       Base Stats      HP  Attack  Special Attack   Speed  Defense  \
count      1025.0  1025.0  1025.0          1025.0  1025.0   1025.0
mean        428.0    70.0    78.0            70.0    67.0     73.0
std         113.0    27.0    30.0            30.0    29.0     29.0
min         175.0     1.0     5.0            10.0     5.0      5.0
25%         323.0    50.0    55.0            47.0    45.0     50.0
50%         450.0    68.0    75.0            65.0    65.0     70.0
75%         508.0    85.0   100.0            90.0    88.0     90.0
max         720.0   255.0   181.0           173.0   200.0    230.0

       Special Defense
count           1025.0
mean              70.0
std               27.0
min               20.0
25%               50.0
50%               67.0
75%               86.0
max              230.0
```

[ ]:
```
import plotly.graph_objects as graph_objects
from plotly.subplots import make_subplots
import plotly.express as px
```

[ ]:
```
import plotly.graph_objects as graph_objects
from plotly.subplots import make_subplots
import plotly.express as px
```

[ ]:
```
# customise pie chart colours
pie_chart_colours = [
    '#E64556',  # HP (dark pink)
    '#A2D7D5',  # Attack (light teal)
    '#55A3AB',  # Special Attack (dark teal)
    '#7D78A3',  # Speed (purple)
    '#FCC499',  # Defense (ligh orange)
    '#FAAA6D'   # Special Defense (dark orange)
]

# Number of rows and columns in the subplot grid
n_types = len(type_df_mean_dict)
# Adjusting to 3 columns
n_cols = 3
# There are 18 types (18/3=6)
```

```python
n_rows = 6

# Create subplot grid using Plotly
fig = make_subplots(
    rows=n_rows, cols=n_cols,
    subplot_titles=[type_name.replace("df_type_", "").replace("_mean", "") for␣
 ↪type_name in type_df_mean_dict.keys()],
    specs=[[{'type': 'pie'}] * n_cols for _ in range(n_rows)]
)

# Loop through each DataFrame in type_df_mean_dict and create a pie chart in␣
 ↪each subplot
for i, (type_name, type_df) in enumerate(type_df_mean_dict.items()):
    # Extract the the different stats
    stats = type_df[all_stats].values.flatten()

    # Create labels for the stats
    labels = all_stats

    # Calculate row and column for each pie chart
    row = i // n_cols + 1
    col = i % n_cols + 1

    # Create the pie chart for the current type
    pie = graph_objects.Pie(
        labels=labels,
        values=stats,
        name=type_name.replace("df_type_", "").replace("_mean", ""),
        hole=0.3,
        marker=dict(colors=pie_chart_colours)
        )

    # Add the pie chart to the appropriate subplot
    fig.add_trace(pie, row=row, col=col)

# Update the layout
fig.update_layout(
    title_text="Base Stats Composition by Pokémon Type",
    title_font=dict(
        family='Impact',
        color='#042C3B', #navy
        size=28
    ),
    height=2000,
    width=1000,
    title_x=0.5,  # Center the title
    legend=dict(
```

```
        # positioning legend
        x=-0.08,
        y=1.05,
        # setting background colour of legend transparent
        bgcolor='rgba(0,0,0,0)'
    )
)



# Show the plot
fig.show()
```

[ ]:
```
# Set up the subplot grid (2 rows x 3 columns)
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
# Flatten so we can loop easily
axes = axes.flatten()
fig.suptitle('Impact of Individual Stats on Base Stats', **impact_font,␣
 ↪size=25, y=0.985)

# Loop through each stat and plot on its own subplot
for idx, stat in enumerate(all_stats):
    ax = axes[idx]

    # Scatter plot
    ax.scatter(df_c[stat], df_c['Base Stats'], alpha=0.5)

    # Calculate and add line of best fit
    m, b = np.polyfit(df_c[stat], df_c['Base Stats'], 1)
    ax.plot(df_c[stat], m * df_c[stat] + b, color='red')

    # Customise each subplot
    ax.set_xlabel(stat, fontsize=11, **verdana_font)
    ax.set_title(f'Impact of {stat} on Base Stats', fontsize=15, **verdana_font)
    ax.tick_params(axis='both', which='major', labelsize=9)

    y= df_c['Base Stats']
    y_pred= m * df_c[stat] + b

    # Calculate R²
    tss = np.sum((y - np.mean(y))**2)
    rss = np.sum((y - y_pred)**2)
    r_squared = 1 - (rss / tss)

    # Add title and R² value inside the plot
    ax.set_title(f'{stat} vs Base Stats', fontsize=12)
    ax.set_xlabel(stat, fontsize=10)
    ax.tick_params(axis='both', which='major', labelsize=9)
```

```python
    # Add slope and intercept as a text inside the plot
    textstr = f"Slope: {m:.2f}\nIntercept: {b:.2f}\nR²: {r_squared:.2f}"
    ax.text(0.02, 0.98, textstr, transform=ax.transAxes,
            fontsize=9, verticalalignment='top', **verdana_font)

    if idx % 3 == 0:
        ax.set_ylabel('Base Stats', fontsize=11, **verdana_font)

# Adjust layout and print
plt.tight_layout()
plt.show()
```

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
```

[ ]:
```python
# 1. Define X and y
X = df_c[['Base Stats', 'Egg Cycles']]
y = df_c['Capturing Rate']

# 2. Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# 3. Random Forest Model
model = RandomForestRegressor(n_estimators=200, random_state=67)  # 100 trees␣
↪in the forest

# 4. Fit the model
model.fit(X_train, y_train)

# 5. Predictions and score
y_pred = model.predict(X_test)

# 6. Evaluate
print(f"R² Score: {r2_score(y_test, y_pred)}")
```

```
R² Score: 0.9611803650469675
```

[ ]:
```
# 1. Define X and y
X = df_c[['Base Stats', 'Capturing Rate', ]]
y = df_c['Is Legendary']

# 2. Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# 3. Random Forest Model
model = RandomForestRegressor(n_estimators=200, random_state=67)  # 100 trees␣
↪in the forest

# 4. Fit the model
model.fit(X_train, y_train)

# 5. Predictions and score
y_pred = model.predict(X_test)

# 6. Evaluate
print(f"R² Score: {r2_score(y_test, y_pred)}")
```

```
R² Score: 0.8356980469693582
```

[ ]:
```
from causalinference import CausalModel
```

[ ]:
```
model = CausalModel(
    Y=df_c['Base Stats'].values,
    D=df_c['Is Legendary'].values,
    X=df_c[['Total Weakness', 'Height Inches','Weight Pounds', 'Capturing Rate',
    'Egg Cycles', 'Gen'
    ]].values
)
```

```
model.est_via_ols()
model.est_via_matching()

# Visualizing the outcomes
plt.figure(figsize=(12, 6))
plt.suptitle('Effects of Legendary Status on Total Base Stats', fontsize=20,␣
 ↪**impact_font)
plt.subplot(1, 2, 1)
plt.hist(df_c[df_c['Is Legendary'] == 0]['Base Stats'], alpha=0.5,␣
 ↪label='Control', color='#92D1B3')
plt.hist(df_c[df_c['Is Legendary'] == 1]['Base Stats'], alpha=0.5, label='Is␣
 ↪Legendary', color='#B74555')
plt.title('Distribution of Outcomes')
plt.xlabel('Base Stats', **verdana_font)
plt.ylabel('Frequency', **verdana_font)
plt.legend(frameon=False)

plt.subplot(1, 2, 2)
treated_mean = df_c[df_c['Is Legendary'] == 1]['Base Stats'].mean()
control_mean = df_c[df_c['Is Legendary'] == 0]['Base Stats'].mean()
plt.bar(['Control', 'Is Legendary'], [control_mean, treated_mean],␣
 ↪color=['#92D1B3', '#B74555'])
plt.title('Average Base Stats by Group', **verdana_font)
plt.ylabel('Average Base Stats', **verdana_font)

plt.tight_layout()
plt.show()
```