Build a Full ML Pipeline

Due: July 15 (Monday) | **Total Points:** 100
Submission: Share a link to your completed Google Colab notebook (set sharing to "University of Colorado Boulder")

🎯 What You'll Learn

- Practice exploratory data analysis (EDA)
- Build a complete machine learning pipeline
- Build a complete Scikit-learn ML pipeline
- Train and evaluate classification models
- Reflect on your design decisions
- Use AI tools responsibly (if you choose to use them)

🚀 You May Use AI, But...

You may use AI for:

- Ask for help writing functions
- Look up syntax
- Get suggestions on model types

You must NOT:

- Submit copy-pasted code without understanding it
- Skip explanation and reflection questions

📊 Dataset Choices

Choose **any** dataset that interests you from Kaggle. Make sure it contains **both numerical and categorical columns** plus a **target variable** you'd like to predict.

Part 1 – Explore the Data (20 pts)

1. Load the dataset with **Pandas**
2. Use `.head()`, `.info()`, `.describe()`, and `.isnull().sum()` and any other commands to explore the dataset
3. Create visualizations to show distributions & correlations
4. **Markdown #** Explain:
   - What do the rows represent?
   - What's your **target** variable?
   - What do you think each feature means?

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as plt
import seaborn as sns
```

```
Mounted at /content/drive
```

```
---------------------------------------------------------------------------
                                  Traceback (most recent call last)
NameError: name 'drive' is not defined
```

```
drive.mount('/content/drive', force_remount=True)
```

```
import pandas as pd
```

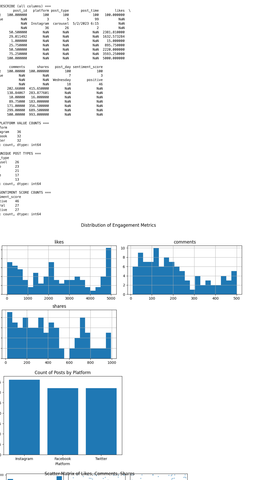
```
Mounted at /content/drive
```

```
# # Part 1 Code Cell:
file_path = '/content/drive/MyDrive/ucla_midwa/datasets/social_media_engagement2.csv'

df = pd.read_csv(file_path)

print("=== HEAD ===")
print(df.head(), "\n")

print("=== INFO ===")
df.info()
print()

print("=== DESCRIBE (all columns) ===")
print(df.describe(include='all'), "\n")

print("=== PLATFORM VALUE COUNTS ===")
print(df['platform'].value_counts(), "\n")

print("=== UNIQUE POST TYPES ===")
print(df['post_type'].unique(), "\n")

print("=== SENTIMENT SCORE COUNTS ===")
print(df['sentiment_score'].value_counts(), "\n")
```

```
df['likes','comments','shares'].hist(figsize=(8,8))
plt.suptitle('Distribution of Engagement Metrics')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

plt.figure(figsize=(6,4))
counts = df['platform'].value_counts()
plt.bar(counts.index, counts.values)
plt.title('Count of Posts by Platform')
plt.xlabel('Platform')
plt.ylabel('Number of Posts')
plt.show()
```

```
scatter_matrix(df[['likes','comments','shares']], figsize=(8,8))
plt.suptitle('Scatter-Matrix of Likes, Comments, Shares', y=0.95)
plt.tight_layout()
plt.show()
```

```
=== HEAD ===
   post_id  platform post_type  ... likes  comments  shares
0  P000001  Twitter      image  ...  8373       1011     454
1  P000002  Facebook   carousel  ...  5/14/2023 8:45  3668     432     694
2  P000003  Facebook   carousel  ...  5/15/2023 8:15  3338     504     666
3  P000004   Twitter      image  ...  5/16/2023 8:30  1161      98     102
4  P000005   Twitter      video  ...  5/21/2023 8:30  3499     247     286

  post_day sentiment_score
0   Thursday       positive
1     Monday       positive
2    Tuesday       negative
3   Thursday       negative
4     Sunday       positive

=== INFO ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   post_id         180 non-null    int64
 1   platform        180 non-null    object
 2   post_type       180 non-null    object
 3   post_time       180 non-null    object
 4   likes           180 non-null    int64
 5   comments        180 non-null    int64
 6   shares          180 non-null    int64
 7   post_day        180 non-null    object
 8   sentiment_score 180 non-null    object
dtypes: int64(4), object(5)
memory usage: 12.8+ KB

=== DESCRIBE (all columns) ===
            post_id platform post_type  ...        likes     comments       shares
count    180.000000      180       180  ...   180.000000   180.000000   180.000000
unique          NaN        4         4  ...          NaN          NaN          NaN
top             NaN  Twitter     image  ...          NaN          NaN          NaN
freq            NaN       50        50  ...          NaN          NaN          NaN
mean     P000090.500      NaN       NaN  ...  3422.111111   499.177778   499.177778
std       P000052.09      NaN       NaN  ...  1421.111556   344.111556   344.111556
min      P000001.000      NaN       NaN  ...   500.000000    10.000000    10.000000
25%      P000045.750      NaN       NaN  ...  2250.000000   200.000000   200.000000
50%      P000090.500      NaN       NaN  ...  3499.000000   247.000000   286.000000
75%      P000135.250      NaN       NaN  ...  4500.000000   800.000000   800.000000
max      P000180.000      NaN       NaN  ...  5000.000000  1500.000000  1500.000000

            comments       shares post_day sentiment_score
count     180.000000   180.000000      180            180
unique           NaN          NaN        7              3
top              NaN          NaN  Wednesday       positive
freq             NaN          NaN       30             90
mean      499.177778   499.177778      NaN            NaN
std       344.111556   344.111556      NaN            NaN
min        10.000000    10.000000      NaN            NaN
25%       200.000000   200.000000      NaN            NaN
50%       247.000000   286.000000      NaN            NaN
75%       800.000000   800.000000      NaN            NaN
max      1500.000000  1500.000000      NaN            NaN

=== PLATFORM VALUE COUNTS ===
platform
Instagram    56
Facebook     52
Twitter      32
Name: count, dtype: int64

=== UNIQUE POST TYPES ===
['image' 'carousel' 'video' 'poll']

=== SENTIMENT SCORE COUNTS ===
sentiment_score
positive    90
neutral     63
negative    27
Name: count, dtype: int64
```



🔧 Your Explanation (Part 1)

Each row represents a single social media post, identified by post_id. The other columns represent attributes of that post (platform, time, engagement, etc.)

I'll create a regression model of engagement, I am using likes as my target variable

Features:

- post_id: Unique identifier
- platform: Social Media Platform
- post_type: Type of post
- post_time: Timestamp of when post went live
- likes: Number of "likes" (target variable)
- comments: Number of comments
- shares: Number of times post was shared
- post_day: day of the week
- sentiment_score: categorical: positive/neutral/negative

Part 2 – Handle Missing Data (Extra credit 10 pts)

1. Identify columns with missing values
2. Decide on an imputation strategy (**drop / 0 / mean / median / mode / 'Unknown'**)
3. Apply using **Pandas** or **SimpleImputer**
4. **Markdown #** Explain **why** you chose each strategy.

```
# # Part 2 Code Cell:
# Your code here
from sklearn.impute import SimpleImputer

print("Missing values before imputation:")
print(df.isnull().sum(), "\n")

#Missing values before imputation:
post_id         0
platform        0
post_type       0
post_time       0
likes           0
comments        0
shares          0
post_day        0
sentiment_score 0
dtype: int64
```

🔧 Explanation of Imputation Choices

Because every column reports 0 missing values, there is nothing to impute or drop. The dataset is complete.

---

bold text

Part 3 – Prepare Features (20 pts)

1. Identify which features are **numeric** vs **categorical**
2. Decide whether to scale (`StandardScaler`, etc.) or **encode** (`OneHotEncoder`) each one
3. Build a `ColumnTransformer` applying the chosen preprocessing
4. **Markdown #** Why scale? Why encode? Which features did you leave unchanged?

```
# # Part 3 Code Cell:
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# 1. Define feature lists
numeric_features = ['likes', 'comments', 'shares']
categorical_features = ['platform', 'post_type', 'post_day']

# 2. Build ColumnTransformer (name OneHot dense directly)
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ],
    remainder='drop'
)

# 3. Fit & transform
preprocessor.fit(df)
X_transformed = preprocessor.transform(df)  # now this is a CSR × 21 numpy array

# 4. Grab the correct output names
feature_names = preprocessor.get_feature_names_out()

df_transformed = pd.DataFrame(
    X_transformed,
    columns=feature_names,
    index=df.index
)

print("Original shape:", df.shape)
print("Transformed shape:", df_transformed.shape)
print(df_transformed.columns.tolist())  # confirm you have 21 column names
```

```
Original shape: (180, 9)
Transformed shape: (180, 21)
['num__likes', 'num__comments', 'num__shares', 'cat__platform_Facebook', 'cat__platform_Instagram', 'cat__platform_Twitter', 'cat__post_type_carousel', 'cat__post_type_image', 'cat__post_type_poll', 'cat__post_type_text', 'cat__post_type_video', 'cat__post_day_Friday', 'cat__post_day_Monday', 'cat__post_day_Saturday', 'cat__post_day_Sunday', 'cat__post_day_Thursday', 'cat__post_day_Tuesday', 'cat__post_day_Wednesday']
```

🔧 Feature-Prep Rationale

Explain why you chose to scale/encode / leave alone each feature...

- scaled the numeric features = ['likes', 'comments', 'shares'] because the three engagement counts have different ranges and distributions. So we are creating a zero-mean space to prevent any one metric from dominating the model because of its scale.
- encoded categorical_features = ['platform', 'post_type', 'post_day', 'sentiment_score'] because these were string objects and Machine Learning can't work with strings so we converted them over to binary/numeric variable so the model can differentiate like "Facebook" from "Instagram"
- dropped post_id and post_time. post_id was only an identifier and didn't add anything to the prediction, and post_time would be better handled via explicit datetime features if needed

Everything else was either scaled or encoded

Part 4 – Build a Pipeline & Train a Model (20 pts)

1. Wrap **preprocessing** + **model** in a `Pipeline`
2. Split data using `train_test_split`
3. Fit the pipeline and evaluate accuracy
4. Print the final **test accuracy**

```
from sklearn.model_selection import train_test_split
# Part 4 Code Cell:
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
# Print the final test accuracy

df = pd.read_csv(file_path)

# Select features & target
X = df[['platform', 'post_type', 'shares']]  # numeric
X_cat = df[['platform', 'post_type', 'post_day']] # categorical
y = df['sentiment_score']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_cat, y, test_size=0.2,
    random_state=42,
    stratify=y
)

# Build numeric features
scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_tr)
X_te_num = scaler.transform(X_te)

# One-hot encode categorical features
ohe = OneHotEncoder(handle_unknown='ignore')
X_tr_cat_enc = ohe.fit_transform(X_tr_cat).toarray()
X_te_cat_enc = ohe.transform(X_te_cat).toarray()

# Concatenate back together
X_tr_full = np.hstack([X_tr_scaled, X_tr_cat_enc])
X_te_full = np.hstack([X_te_scaled, X_te_cat_enc])

# Fit the model
model = LogisticRegression(max_iter=1000)
model.fit(X_tr_full, y_tr)

# Predict & evaluate
y_pred = model.predict(X_te_full)
acc = accuracy_score(y_te, y_pred)
print("Test accuracy:", round(acc, 3))
```

```
Test accuracy: 0.45
```

Part 5 – Try a Different Model (10 pts)

1. Try a different model (feel free to look up things on the internet)
2. Train & evaluate
3. **Markdown #** Which did better?

```
# # Part 5 Code Cell:
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

df = pd.read_csv(file_path)
X_cat = df[['likes', 'comments', 'shares']]  # numeric
X_cat = df[['platform', 'post_type', 'post_day']] # categorical
y = df['sentiment_score']

X = pd.concat([X_num, X_cat], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,
    y, test_size=0.2, random_state=42, stratify=y
)

# Build preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ]
)

rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

rf_pipeline.fit(X_train, y_train)
rf_acc = accuracy_score(y_test, rf_pipeline.predict(X_test))

results_df = pd.DataFrame([
    {'Model': 'Logistic Regression', 'Test Accuracy': acc},
    {'Model': 'Random Forest', 'Test Accuracy': rf_acc}
])
results_df
```

	Model	Test Accuracy
0	Logistic Regression	0.45
1	Random Forest	0.45

🔧 Model Comparison Discussion

Discuss which model performed better

I noticed that both models gave the same accuracy score and when asking AI, the answer I found is that when both a high bias and high capacity model converge to the same error, we have hit the limit of what the features can predict. I would need more discriminative features.

Part 6 – Manual Code Walkthrough (10 pts)

🔧 Rewrite your **final pipeline** in markdown to show your understanding of every step.

Example:

- Step 1: reading the dataframe
- Step 2: define features (X) and target (Y)
- Step 3: specify which columns are numeric and categorical
- Step 4: build a column transform for preprocessing
- Step 5: wrap preprocessing and the model in a pipeline
- Step 6: split into train and test sets
- Step 7: fit the pipeline on the training data
- Step 8: predict on the test features
- Step 9: evaluate final test accuracy

Part 7 – Reflections + AI Use Disclosure (20 pts - 250-300 words)

Answer all questions below in Markdown:

1. **Problem statement:** What question were you trying to answer?
2. **Approach:** Briefly describe this exploration, feature prep, and model selection.
3. **Results:** How did your model perform? Was it good enough?
4. **Learning:** What was most surprising about this project?
5. **AI usage:** If you used AI tools, how exactly did you use them, and how did you validate the output?

1. Problem Statement: We were looking to build an ML pipeline that takes numerical and categorical features to predict a target class.
2. Approach:

- Explore the data set
- Handling missing data
- Engineering and selecting categorical variables, scaling, one hot encoding, and column transform
- Training and testing a baseline model and a more complex model
- Tuning hyperparameters to compare performance

3. Challenges: I think I am having a disconnect between what we are doing and the math. I understand things better when I can somehow visualize them in the real world, and I am struggling to connect the math to the code. I think more
4. Learning: I learned that you can't just create an automated process that allows engineers to refine existing models alongside constant performance validations.
5. AI usage: I used AI tools to try and explain the math more and give me examples to visualize the pipeline in the real world. I used it for filling in concepts I missed in class and for debugging my code.