Data Analytics in Python and SQL: Final Project

Guillermo Casillas, Valeria Chavez, Benjamin Catton

Background Information

1. Two Major Datasets: IPO and Funding Rounds
2. Impact of Investor Behavior and Market Trends
3. Compare the path from startup to IPO
4. VC Firms, startup founder, PE, and analysts

Set-Up

```
[5]: fr_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52928 entries, 0 to 52927
Data columns (total 23 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       52928 non-null  int64
 1   funding_round_id         52928 non-null  int64
 2   object_id                52928 non-null  object
 3   funded_at                52680 non-null  object
 4   funding_round_type       52928 non-null  object
 5   funding_round_code       52928 non-null  object
 6   raised_amount_usd        52928 non-null  float64
 7   raised_amount            52928 non-null  float64
 8   raised_currency_code     49862 non-null  object
 9   pre_money_valuation_usd  52928 non-null  float64
 10  pre_money_valuation      52928 non-null  float64
 11  pre_money_currency_code  26883 non-null  object
 12  post_money_valuation_usd 52928 non-null  float64
 13  post_money_valuation     52928 non-null  float64
 14  post_money_currency_code 30448 non-null  object
 15  participants             52928 non-null  int64
 16  is_first_round           52928 non-null  int64
 17  is_last_round            52928 non-null  int64
 18  source_url               40382 non-null  object
 19  source_description       43439 non-null  object
 20  created_by               48291 non-null  object
 21  created_at               52928 non-null  object
 22  updated_at               52928 non-null  object
dtypes: float64(6), int64(5), object(12)
memory usage: 9.3+ MB
```

Summary

- Set-Up pandas and imported our CSV files
- After we ran a quick .info() to see the variables we were working with and their types

Data Cleansing

```python
# First, create the IPO label column
ipo_df['went_ipo'] = 1  # Mark IPO companies

# Merge: left join from funding to IPO to label companies
merged_df = fr_df.merge(
    ipo_df[['object_id', 'went_ipo']],
    on='object_id',
    how='left'
)

# Fill NaN with 0 → companies that didn't IPO
merged_df['went_ipo'] = merged_df['went_ipo'].fillna(0).astype(int)
```

```python
# Grouping and aggregating per startup (object_id)
company_agg = merged_df.groupby('object_id').agg({
    'funding_round_id': 'count',                   # total funding rounds
    'raised_amount_usd': ['sum', 'mean', 'max'],   # investment stats
    'participants': ['sum', 'mean', 'max'],        # participant stats
    'is_first_round': 'sum',                       # how often it was the first round
    'is_last_round': 'sum',                        # how often it was the last round
    'went_ipo': 'max'                              # target variable (still per company)
})

# Flatten multi-level columns
company_agg.columns = ['_'.join(col).strip() for col in company_agg.columns.values]

# Reset index so object_id becomes a column again
company_agg = company_agg.reset_index()

# Preview result
company_agg.head()
```

Summary

- Adds a binary IPO label to the IPO dataset and merges IPO info into the funding dataset by company ID.
- Marks missing IPOs as 0 (did not IPO).
- Converts the label to integer (0 or 1).

- Groups data by company.
- Aggregates funding stats (count, sum, mean, max).
- Counts first/last rounds.
- Flags IPO outcome.

Data Cleansing

```python
# Checking for missing values
missing_values = company_agg.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_values / len(company_agg)) * 100
missing_df = pd.DataFrame({'Missing Count': missing_values, 'Missing %': missing_percentage})
print(missing_df)
```

```python
# Summary stats for key variables

summary_stats = company_agg[['raised_amount_usd_sum', 'participants_sum', 'funding_round_id_count']].describe()

# Format the output using style
summary_stats.style \
    .format("{:,.2f}") \
    .set_caption("Summary Statistics of Key Company Metrics") \
    .set_table_styles([{
        'selector': 'caption',
        'props': [('color', 'black'), ('font-size', '16px'), ('text-align', 'center')]
    }])
```

Summary Statistics of Key Company Metrics

	count	mean	std	min	25%	50%	75%	max
raised_amount_usd_sum	31,939.00	13,170,984.84	67,305,586.21	0.00	200,000.00	1,700,000.00	9,100,000.00	5,700,000,000.00
participants_sum	31,939.00	2.53	4.00	0.00	0.00	1.00	3.00	58.00
funding_round_id_count	31,939.00	1.66	1.20	1.00	1.00	1.00	2.00	15.00

Summary

- Checks for missing values in the dataset.
- Calculates count and percentage of missing values per column.
- Stores results in a summary table.

- Selects key company metrics for summary.
- Generates descriptive statistics (count, mean, std, etc.).
- Formats and styles the summary table for better presentation.

Data Analysis

```
# IPO Outcome Stats Table
ipo_outcome_stats = company_agg.groupby('went_ipo_max')[
    ['raised_amount_usd_sum', 'participants_sum']
].mean().rename(index={0: "Did not IPO", 1: "Went IPO"})

ipo_outcome_stats.style.format("{:,.0f}").set_caption("Average Stats by IPO Outcome")
```

Average Stats by IPO Outcome

went_ipo_max	raised_amount_usd_sum	participants_sum
Did not IPO	11,581,749	3
Went IPO	108,449,318	4

Summary

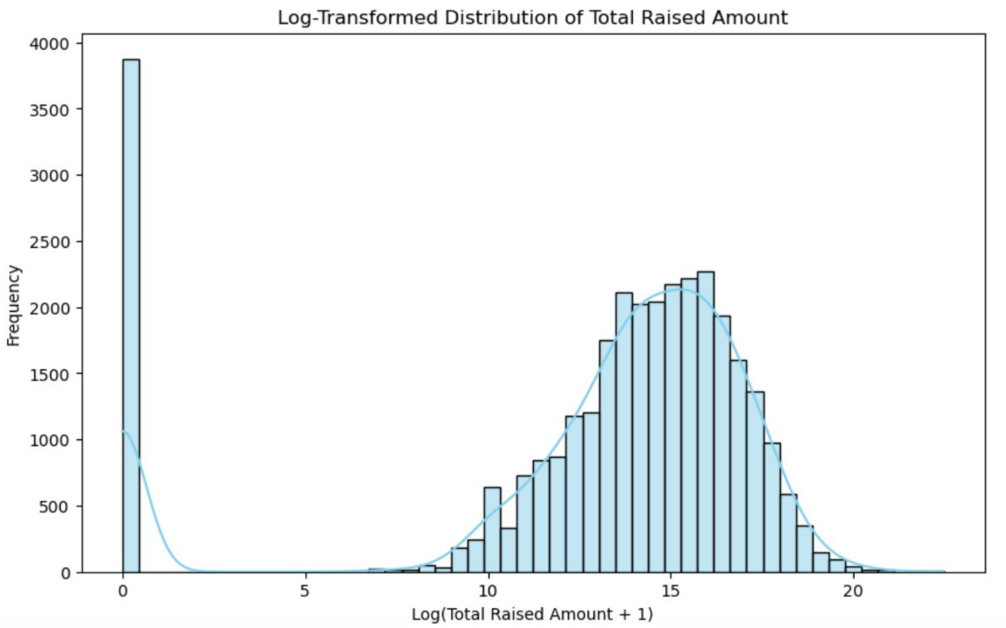
- Groups companies by IPO outcome (IPO vs. non-IPO).
- Calculates average funding and participant count for each group.
- Renames the outcome labels for clarity.

```
# Top 10 Companies by Total Raised
top_raised = company_agg[['object_id', 'raised_amount_usd_sum']].sort_values(
    by='raised_amount_usd_sum', ascending=False).head(10)

top_raised.style.format({"raised_amount_usd_sum": "${:,.0f}"}).set_caption("Top 10 Companies by Total Raised")
```

- Selects company ID and total funding raised.
- Sorts by funding amount in descending order.
- Displays the top 10 highest-funded companies.

Data Analysis

```
# Log distribution of raised amount

plt.figure(figsize=(10,6))
# Add 1 to avoid log(0)
sns.histplot(np.log1p(company_agg['raised_amount_usd_sum']), bins=50, kde=True, color='skyblue')
plt.title('Log-Transformed Distribution of Total Raised Amount')
plt.xlabel('Log(Total Raised Amount + 1)')
plt.ylabel('Frequency')
plt.show()
```

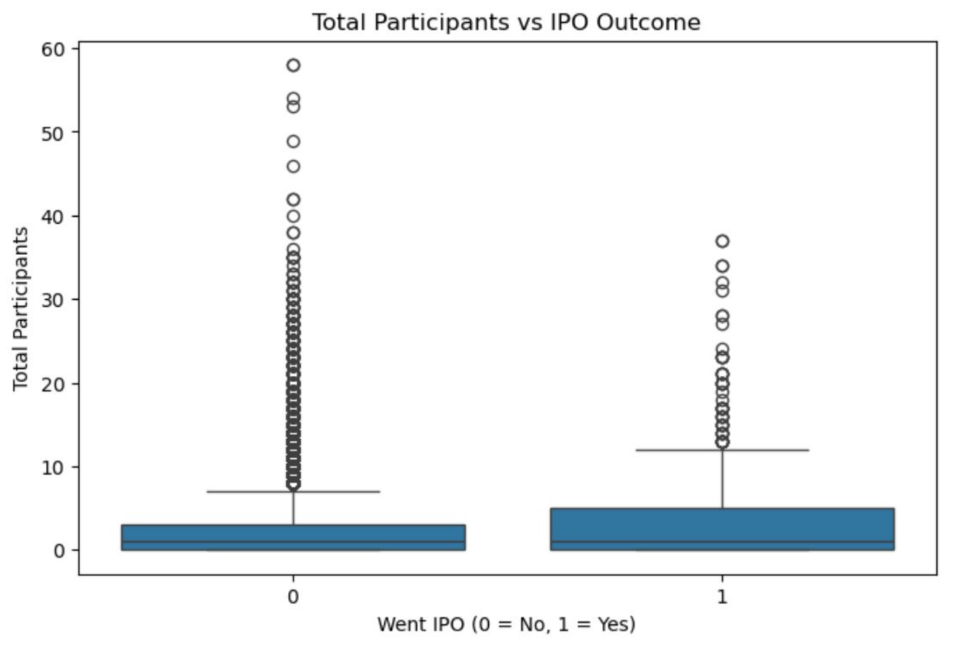


Summary

- Creates a histogram of total funding raised (log-transformed).
- Applies log(raised amount + 1) to handle skew and zero values.
- Uses 50 bins and overlays a KDE (density) curve.

Data Analysis

```python
# Boxplot of Participants vs IPO Status
plt.figure(figsize=(8,5))
sns.boxplot(x='went_ipo_max', y='participants_sum', data=company_agg)
plt.title('Total Participants vs IPO Outcome')
plt.xlabel('Went IPO (0 = No, 1 = Yes)')
plt.ylabel('Total Participants')
plt.show()
```

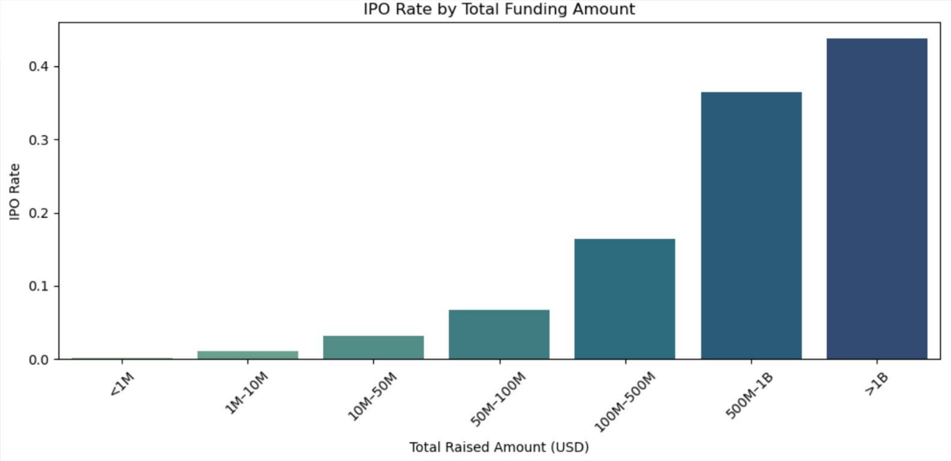


Summary

- Creates a boxplot comparing participant counts by IPO status.
- Groups companies based on whether they went public or not.
- Shows distribution and outliers of total participants per group.

Data Analysis

```
# Funding Raised Bin vs IPO Rate

# Define bins for raised amount (you can adjust bin edges)
bins = [0, 1e6, 1e7, 5e7, 1e8, 5e8, 1e9, company_agg['raised_amount_usd_sum'].max()]
labels = [
    '<1M', '1M–10M', '10M–50M', '50M–100M',
    '100M–500M', '500M–1B', '>1B'
]

# Bin the data
company_agg['raised_amount_bin'] = pd.cut(company_agg['raised_amount_usd_sum'], bins=bins, labels=labels)

# Calculate IPO rate per bin
ipo_by_bin = (
    company_agg.groupby('raised_amount_bin')
    .agg(ipo_rate=('went_ipo_max', 'mean'))
    .reset_index()
    .rename(columns={'ipo_rate': 'IPO Rate'})
)

plt.figure(figsize=(10, 5))
sns.barplot(data=ipo_by_bin, x='raised_amount_bin', y='IPO Rate', palette='crest')
plt.title('IPO Rate by Total Funding Amount')
plt.xlabel('Total Raised Amount (USD)')
plt.ylabel('IPO Rate')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Summary

- Categorizes companies into bins based on total funding raised.
- Assigns each company to its appropriate funding range.
- Computes the average IPO rate for each funding bin.

Data Analysis

```
# Linear regression model

# 1. Preparing the data
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Features and target
X = company_agg[[
    'raised_amount_usd_sum'
]]

y = company_agg['went_ipo_max']

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

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# 2. Train the logistic regresion model

from sklearn.linear_model import LogisticRegression

# Initialize and train
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)
```

```
# 3. Evaluate the model

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Predictions
y_pred = logreg.predict(X_test_scaled)

# Metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9838760175328741
Precision: 0.75
Recall: 0.02857142857142857
F1 Score: 0.05504587155963303

Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      6283
           1       0.75      0.03      0.06       105

    accuracy                           0.98      6388
   macro avg       0.87      0.51      0.52      6388
weighted avg       0.98      0.98      0.98      6388
```

Summary

- Defines input features and IPO outcome.
- Scales features for better model performance.

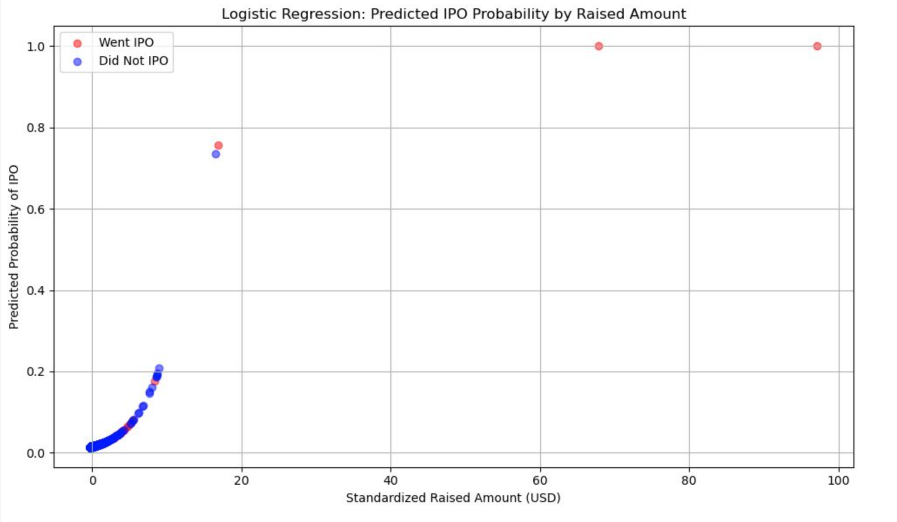
Splits data into training and testing sets.

Trains a logistic regression model to predict IPO likelihood.

Generates predictions and prediction probabilities.

Data Analysis

```
# Predicted probability of IPO (class 1)
prob_prediction = logreg.predict_proba(X_test_scaled)[:, 1]

# Mask for actual IPO vs not
ipo_mask = y_test.values == 1
non_ipo_mask = ~ipo_mask

# Plot
plt.figure(figsize=(10,6))
plt.scatter(X_test_scaled[ipo_mask], prob_prediction[ipo_mask], color='red', label='Went IPO', alpha=0.5)
plt.scatter(X_test_scaled[non_ipo_mask], prob_prediction[non_ipo_mask], color='blue', label='Did Not IPO', alpha=0.5)

plt.xlabel('Standardized Raised Amount (USD)')
plt.ylabel('Predicted Probability of IPO')
plt.title('Logistic Regression: Predicted IPO Probability by Raised Amount')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Summary

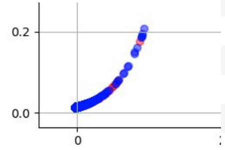
- Calculates and displays the ROC AUC score.
- Plots the ROC curve to evaluate model performance.

SQL Analysis

```python
# SQL Query 4: Group by – Average raise by funding round type
query4 = """
SELECT funding_round_type, AVG(raised_amount_usd) AS avg_raised
FROM funding_rounds
GROUP BY funding_round_type;
"""
q4_result = pd.read_sql(query4, connection)
q4_result
```

	funding_round_type	avg_raised
0	angel	3.056193e+05
1	crowdfunding	1.638457e+06
2	other	1.123907e+07
3	post-ipo	1.694044e+08
4	private-equity	2.502106e+07
5	series-a	5.914058e+06
6	series-b	1.134449e+07
7	series-c+	2.116659e+07
8	venture	8.159983e+06

```python
bel='Went IPO', alpha=0.5)
blue', label='Did Not IPO', alpha=0.5)
```




```python
SELECT
    object_id,
    funding_round_id,
    raised_amount_usd,
    RANK() OVER (PARTITION BY object_id ORDER BY raised_amount_usd DESC) AS raise_rank
FROM funding_rounds;
"""
q5_result = pd.read_sql(query5, connection)
q5_result
```

	object_id	funding_round_id	raised_amount_usd	raise_rank
0	c:1	2312	25000000.0	1
1	c:1	889	9500000.0	2
2	c:1	888	5250000.0	3
3	c:1001	1644	5000000.0	1
4	c:10014	6682	0.0	1
...	...	...	...	...
52923	c:9989	5765	500000.0	2
52924	c:9994	3253	250000.0	1
52925	c:9994	6112	250000.0	1
52926	c:9995	3264	750000.0	1
52927	c:9998	3254	475000.0	1

Summary

- Group by type of funding raised
- Ranking of funding round size within each company

SQL Analysis

```
SELECT object_id
FROM funding_rounds
GROUP BY object_id
HAVING COUNT(*) >3;
"""
q9_result = pd.read_sql(query9, connection)
q9_result
```

```
ent IPO', alpha=0.5)
 label='Did Not IPO', alpha=0.5)
```

	object_id
0	c:10015
1	c:10054
2	c:100844
3	c:1010
4	c:10161
...	...
2371	c:980
2372	c:9836
2373	c:9840
2374	c:9949
2375	c:9972

```
WITH avg_funding AS (
    SELECT object_id, AVG(raised_amount_usd) AS avg_raised
    FROM funding_rounds
    GROUP BY object_id
)
SELECT *
FROM avg_funding
WHERE avg_raised > 10000000;
"""
q11_result = pd.read_sql(query11, connection)
q11_result
```

	object_id	avg_raised
0	c:1	1.325000e+07
1	c:10015	1.361384e+07
2	c:10018	1.100000e+07
3	c:1005	2.000000e+07
4	c:10054	1.735714e+07
...	...	...
4606	c:9891	1.080000e+08
4607	c:98929	2.537820e+07

Summary

- Subquery to find companies with more than 3 funding rounds
- The outer query filters based on subquery aggregation

- Use a CTE to calculate average funding per company
- Then select only those companies whose average raise exceeds $10M

Thanks for participating

Conclusion

1

Challenges

Data Merging Complexity:
Matching records across datasets required careful alignment on unique identifiers which can be inconsistent or missing in real-world data.Complex

2

Challenges

Skewed Distributions
Variables like total funding raised were highly skewed, requiring log transformation to support meaningful visualizations and modeling.

3

Suggestions

Add market conditions at time of funding IPO

Timing of funding rounds (years between) to capture growth

4

Suggestions

Interactive dashboards using Tableau to explore IPO trends and company features

Crunchbase API and Pitchbook

Thank you

Questions?