Final Project: Group 3

CREDIT CARD FRAUD DETECTION & AMAZON REVIEW NLP

Neural Networks +

William Aromando, Dan Becker, Joanne Donohue,
Vivin Rajagopalan, and Chace Snedecor

PROJECT OVERVIEW

1. In the first dataset we utilize **simple transaction data** to make predictions surrounding **credit card fraud**

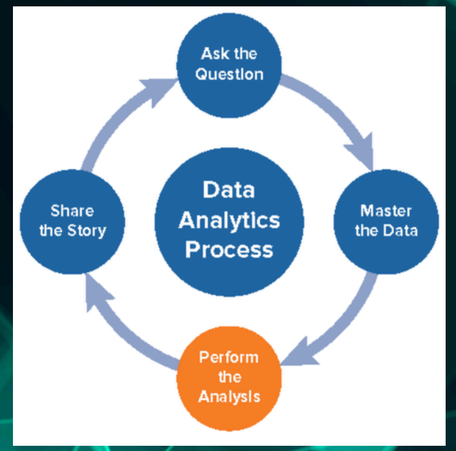
2. The second dataset we utilize **PCA transformation** data from European cardholders to **detect fraud**.

3. The **amazon review dataset** assigns a **sentiment score** whether positive or negative to an amazon review. Making customers more trusting in their purchase decisions.

DATA ANALYSIS

Four Types of Data Analytics

1. *Descriptive Analytics* - What happened? What is happening?
2. *Diagnostic Analytics* - Why did it happen? What are the reasons for past results?
3. *Predictive Analytics* - Will it happen in the future?
4. *Prescriptive Analytics* - What should we do, based on what we expect will happen?



When analyzing financial transaction for Check Fraud,
Techniques for Identifying Anomalies/Outliers

Sequence checks and sequence analysis
Duplicate transactions
Benford's Law

AMPS MODEL:

A - Ask the Questions
M - Master the Data
P - Perform the Analysis
S - Share the Story

Framework -
Data Analytics process

The Analytics Mindset
Ernst & Young Foundation

DIAGNOSTIC ANALYTICS

Diagnostic Analytics

Why did it happen? What are the reasons for past results?

Sequence checks and sequence analysis

Are any checks or purchase orders missing?

Duplicate transactions

Why are there duplicates of some transactions?

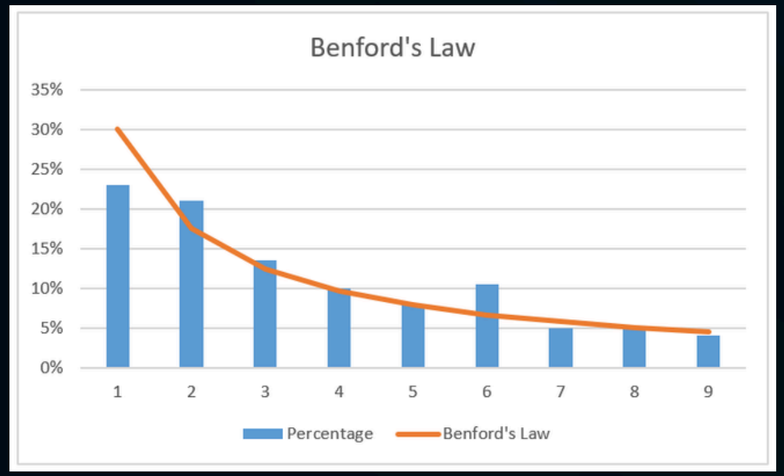
Benford's Law

Is the distribution of the first digit of a dataset different than expected?

EXHIBIT 7.12 Benford Digit Frequency Occurrence

Digit	1st	2nd	3rd	4th
0		12.0%	10.2%	10.0%
1	30.1%	11.4%	10.1%	10.0%
2	17.6%	10.9%	10.1%	10.0%
3	12.5%	10.4%	10.1%	10.0%
4	9.7%	10.0%	10.0%	10.0%
5	7.9%	9.7%	10.0%	10.0%
6	6.7%	9.3%	10.0%	10.0%
7	5.8%	9.0%	9.9%	10.0%
8	5.1%	8.8%	9.9%	10.0%
9	4.6%	8.5%	9.8%	10.0%

Source: Using Benford's Law to Detect Fraud, Association of Certified Fraud Examiners, 2018, https://www.acfe.com/uploadedFiles/Shared_Content/Products/Self-Study_CPE/Using%20Benford's%20Law_2018_final_extract.pdf, accessed March 2019



PREDICTIVE ANALYTICS

Predictive Analytics

Will it happen in the future?

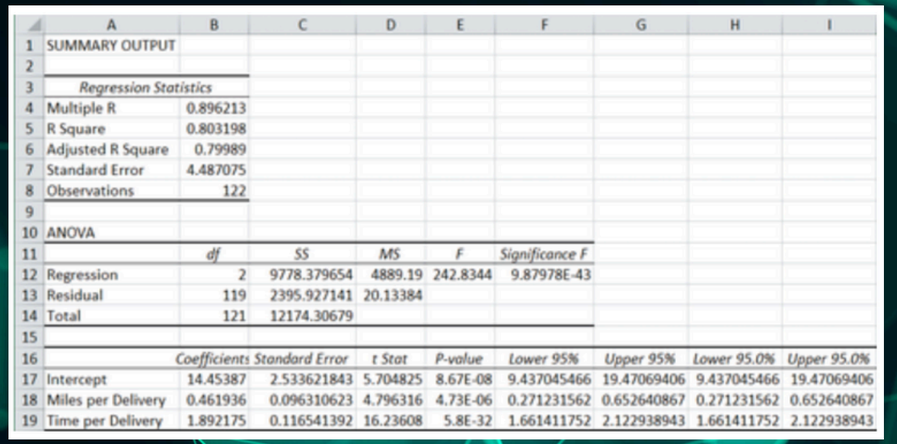
1. Firm likley committed financial statement fraud.
2. Firm did not commit financial statement fraud.

Examining the possibility of occurrence of Check Fraud as it relates to other financial records/reports.

Regression Analysis

Predictive analytics technique used to estimate a specific dependent variable outcome based on independent variable inputs.

Regression Analysis



	A	B	C	D	E	F	G	H	I
1	SUMMARY OUTPUT								
2									
3	*Regression Statistics*								
4	Multiple R	0.896213							
5	R Square	0.803198							
6	Adjusted R Square	0.79989							
7	Standard Error	4.487075							
8	Observations	122							
9									
10	ANOVA								
11		*df*	*SS*	*MS*	*F*	*Significance F*			
12	Regression	2	9778.379654	4889.19	242.8344	9.87978E-43			
13	Residual	119	2395.927141	20.13384					
14	Total	121	12174.30679						
15									
16		*Coefficients*	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
17	Intercept	14.45387	2.533621843	5.704825	8.67E-08	9.437045466	19.47069406	9.437045466	19.47069406
18	Miles per Delivery	0.461936	0.096310623	4.796316	4.73E-06	0.271231562	0.652640867	0.271231562	0.652640867
19	Time per Delivery	1.892175	0.116541392	16.23608	5.8E-32	1.661411752	2.122938943	1.661411752	2.122938943

M. D. Beneish (1999) Predictive Analytics from 1982 to 1992

Eight factors to be predictive of financial statement fraud.

1. Increase in receivables as compared to last period.
2. Decline in gross margin as a percent of sales.
3. Decline in asset quality index.
4. Increase in sales growth in current period.
5. Decrease in depreciation expense.
6. Decrease in selling, general, and administrative costs.
7. Increase in debt (leverage).
8. Higher total accrual to total assets.

Using an Additional Library (PyTorch)

Feature Scaling:

- In the context of the code above, one shall scale the features using the StandardScaler of the scikit-learn library for data normalization before feeding it into the neural network. This shall make the model converge faster and improve the training performance, especially for gradient-based models like the neural network.

Conversion to PyTorch Tensors:

Scaled feature data and target labels are converted to PyTorch tensors with torch.tensor(). According to the architecture of PyTorch models, input data needs to be a tensor

1. Neural Network Model Definition:
   - A basic simple fully connected feedforward network is defined as follows using PyTorch's class nn.Module.
   - The model consists of:
     - Input Layer: The first layer, fc1, absorbs the input features; the number of input features is derived from features_train_tensor.shape[1].
   - Hidden Layers: Two hidden layers followed with ReLU activation functions to introduce nonlinearity and learn more complex patterns.
   - Output Layer: In the output layer, fc3 is just one value that has been passed through a sigmoid activation function in order to predict class probabilities for the classification task (fraud or not).

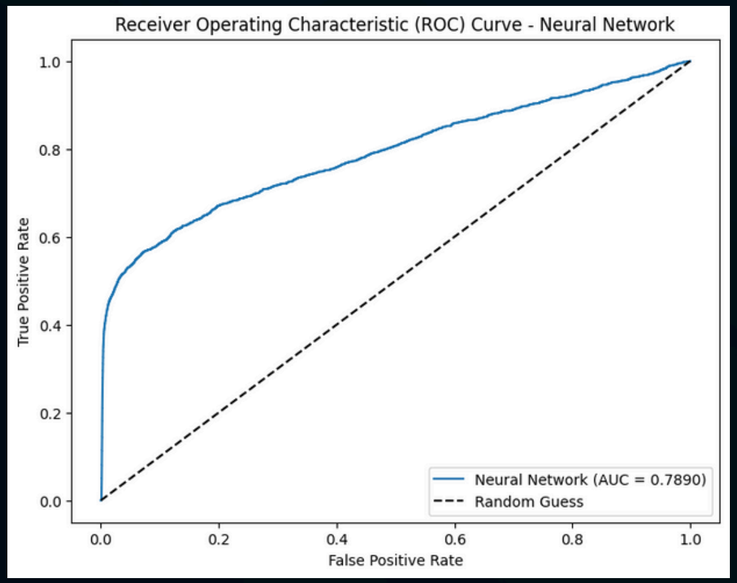
Continuing with Pytorch

Training the Neural Network:

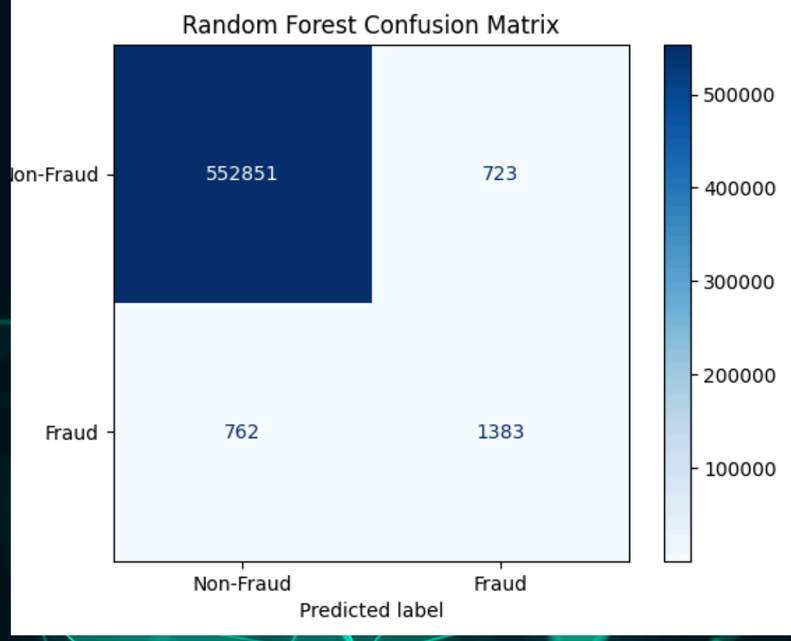
The model will be trained using the Adam optimizer with binary cross-entropy loss, which is the normal loss function for a classification problem where there are two classes.

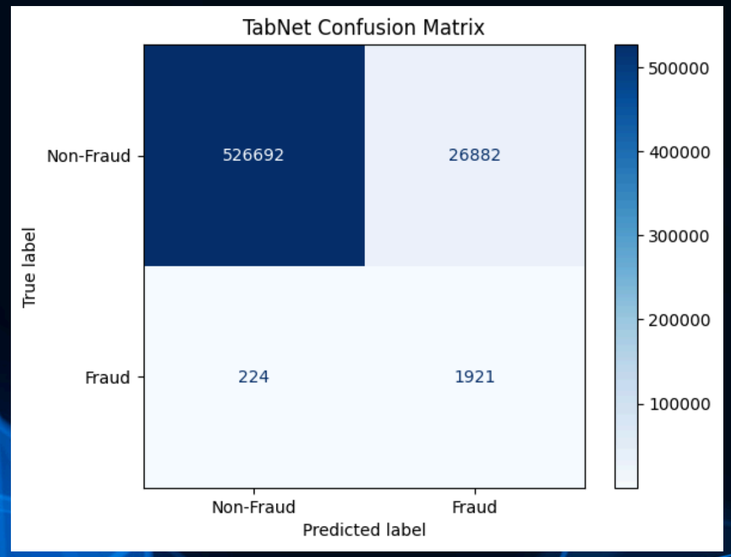
It then iterates over the dataset with a training loop for a fixed number of epochs, here 10. At each epoch, the model performs a forward pass, computes the loss of a particular batch, and backpropagation updates its weights accordingly (loss.backward(), optimizer.step()).

1. Evaluation of the Neural Network:
   - After a sufficient number of epochs, the model performance will be calculated on the test set, features_test_tensor.
- It makes predictions and converts the continuous output into binary labels (0 or 1), using a threshold of 0.5 (predictions_nn >= 0.5).
- Accuracy, precision, recall, F1-score, and AUC-ROC are evaluation metrics to determine how well the model performs in comparison with the test data



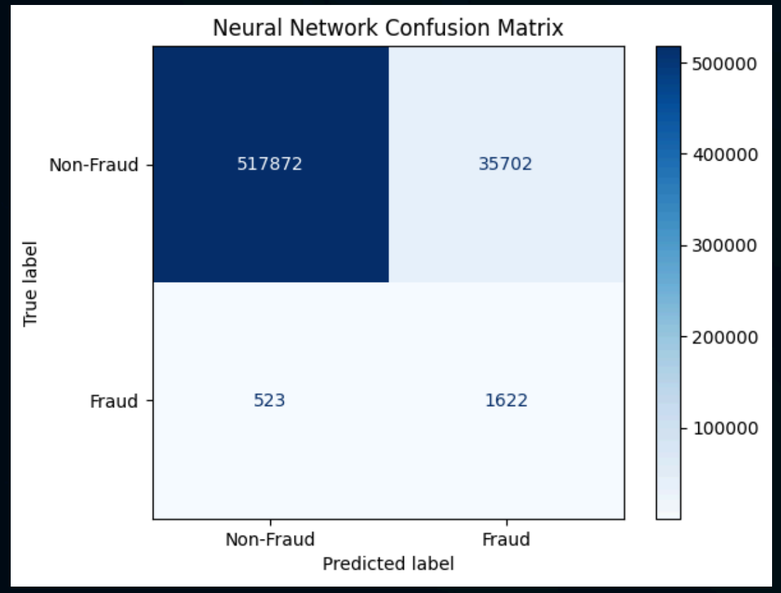
1. Random Forest Confusion Matrix:
   - ConfusionMatrixDisplay.from_predictions is used to compare the true labels (target_test) and the predicted labels from the Random Forest model (rf_predictions).
   - The matrix shows how many instances were correctly or incorrectly classified as either "Non-Fraud" or "Fraud."





1. **TABNET CONFUSION MATRIX:**
   - **SIMILARLY, THE CONFUSION MATRIX FOR THE TABNET MODEL IS GENERATED USING THE TRUE LABELS (TARGET_TEST) AND THE PREDICTIONS FROM THE TABNET MODEL (TABNET_PREDICTIONS).**

1. **NEURAL NETWORK CONFUSION MATRIX:**
   - **FOR THE NEURAL NETWORK, THE PREDICTED LABELS ARE OBTAINED AFTER APPLYING A THRESHOLD TO THE MODEL'S CONTINUOUS OUTPUTS. THESE PREDICTIONS ARE COMPARED WITH THE TRUE LABELS (TARGET_TEST) TO FORM THE CONFUSION MATRIX.**



2

PCA FRAUD DATA APPROACH

With only **492 frauds** in a total of 284,807 transactions (0.172%), the dataset presents a highly unbalanced classification challenge.

Evaluating models based on traditional accuracy metrics would be misleading.

Instead, we'll emphasize on a balanced sample and avoiding overfitting to assess model performance effectively.

PCA TRANSFORMATION

Principal Component Analysis Transformation or (PCA) are new features made by combining the originals in a way that pinpoints the most important patterns or differences in the data.

In our data we have **28 variables** that utilize PCA, for instance V1 in our dataset could represent **"*high transaction dollar amount*" and "*multiple transactions in a short time frame*"** which are potential fraud behaviors.

RAW DATA PREP

	Time	V1	V2	V3	V4	V5	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

284807 rows × 31 columns

- Normalized Time from 0 to 1 to avoid skewing results with time of day
- Robust Scaler to normalize outliers in Amount
- Class is 0 = Not Fraud or 1 = Fraud
- **Major Watch Out: Unbalanced Dataset - biased model**
  - **99% of trx are NOT FRAUD**

```
cc_df['Class'].value_counts()
```

Class	count
0	284315
1	492

dtype: int64

Inital Results

- **Precision = % predicted fraud actually fraud**
- **Recall = % actual fraud cases correctly IDed**
- **F1 = mean of precision and recall**
- **Support = # Fraud cases in test dataset**

- Not Fraud results are all 100% due to model skew

	Class	Precision	Recall	F1-Score	Support
Raw Data (all Not Fraud 100%)					
Option 1: Logistic Regression	Fraud	83%	56%	67%	36
Option 2: Shallow Neural Network	Fraud	75%	75%	75%	36
Option 3: RandomForest	Fraud	77%	47%	59%	36
Option 4: Gradient Boost	Fraud	67%	67%	67%	36
Option 5: Linear SVC	Fraud	7%	97%	14%	36

- **Best Results are through Linear SVC (Option 5):** High recall achieved at cost of extremely low precision, making it impractical for fraud detection (**too many false positives**).
- The **Random Forest model** (Option 3) strikes a balance between precision (77%) and recall (47%), but the **F1-score is still low** (59%)

Inital Results

- **Precision = % predicted fraud actually fraud**
- **Recall = % actual fraud cases correctly IDed**
- **F1 = mean of precision and recall**
- **Support = # Fraud cases in test dataset**

	Class	Precision	Score	Support
Raw Data (all Not Fraud 100%)				
Option 1: Logistic Regression	Fraud	83%	%	36
Option 2: Shallow Neural Network	Fraud	75%	%	36
Option 3: RandomForest	Fraud	77%	%	36
Option 4: Gradient Boost	Fraud	67%	7%	36
Option 5: Linear SVC	Fraud	7%	14%	36

- **Best Results are through Linear SVC (Option 5):** High recall achieved at cost of extremely low precision, making it impractical for fraud detection (**too many false positives**).
- The **Random Forest model** (Option 3) strikes a balance between precision (77%) and recall (47%), but the **F1-score is still low** (59%)

Adjustment 1: Balance Class Records

```
[ ] not_frauds = new_df.query('Class == 0')
    frauds = new_df.query('Class == 1')
    not_frauds['Class'].value_counts(), frauds['Class'].value_counts()
```

```
(Class
 0    284315
 Name: count, dtype: int64,
 Class
 1    492
 Name: count, dtype: int64)
```

```
# Make a new dataset to randomly balance the dataset, 492 samples in each group of Fraud and Not Fraud

balanced_df = pd.concat([frauds, not_frauds.sample(len(frauds), random_state = 1)])
balanced_df['Class'].value_counts()
```

Class	count
1	492
0	492

dtype: int64


```
[ ] balanced_df = balanced_df.sample(frac = 1, random_state = 1)
    balanced_df
```

	12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	
	138	0.694015	-1.553956	-1.682784	0.430354	1.005715	-2.050401	-0.146696	-1.033500	2.493579	0.320829	-0.535481	0.4
	104	-0.263264	-1.112735	-0.540850	1.533411	0.831443	-0.473347	1.190121	0.273799	-0.026055	-0.295255	-0.180459	-0.4
	101	-0.954361	-9.861372	-0.505329	0.269282	0.591319	1.795992	-1.085208	0.354773	0.319261	-0.471379	-0.075890	-0.6
	170	0.685337	-0.056341	-0.326394	0.081963	-0.767352	0.372928	0.445735	-0.091404	0.069401	0.268024	0.261459	0.6
	115	0.959700	0.403010	-0.449314	-0.250583	-0.436450	-0.407617	0.354953	-0.028974	-0.083048	-0.137032	-0.238920	-0.6

- Hack the Sample dataset to be 50:50 Fraud: Not Fraud
- Blend new dataset to randomize cases
- Split into 3 groups: Train, Test and Validate

```
[ ] balanced_df_np = balanced_df.to_numpy()

    x_train_b, y_train_b = balanced_df_np[:700, :-1], balanced_df_np[:700, -1].astype(int)
    x_test_b, y_test_b = balanced_df_np[700:842, :-1], balanced_df_np[700:842, -1].astype(int)
    x_val_b, y_val_b = balanced_df_np[842:, :-1], balanced_df_np[842:, -1].astype(int)

    # check shape to confirm
    x_train_b.shape, y_train_b.shape, x_test_b.shape, y_test_b.shape, x_val_b.shape, y_val_b.shape
```

```
((700, 30), (700,), (142, 30), (142,), (142, 30), (142,))
```

```
[ ] pd.Series(y_train_b).value_counts(), pd.Series(y_test_b).value_counts(), pd.Series(y_val_b).value_counts()
```

```
(1    353
 0    347
 Name: count, dtype: int64,
 0    73
 1    69
 Name: count, dtype: int64,
 0    72
 1    70
 Name: count, dtype: int64)
```

Adjustment 2: Experiment with Hidden NN

Option 7: Shallow Neural Network (Balanced)

- 2 relu
- overfitting?

```
shallow_nn_b = Sequential()
shallow_nn_b.add(InputLayer((x_train.shape[1],)))
shallow_nn_b.add(Dense(2, activation='relu'))
shallow_nn_b.add(BatchNormalization())
shallow_nn_b.add(Dense(1, activation='sigmoid'))

# The filepath needs to end with '.keras' for Keras 3
checkpoint = ModelCheckpoint('shallow_nn_b.keras', save_best_only=True)

shallow_nn_b.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Set epochs up to 40 from 5 because of smaller dataset
shallow_nn_b.fit(x_train_b, y_train_b,validation_data=(x_val_b, y_val_b), epochs = 40, callbacks=[checkpoint])
```

- Experimented with number of hidden ReLu layers in Neural Network
- With 40 epochs, we saw increasing accuracy with train model but LOWER validation accuracy
- This signals OVERFITTING of the model and an overly complicated model, reduce to 1 hidden layer

```
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8473 - loss: 0.3965 - val_accuracy: 0.8662 - val_loss: 0.3960
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8927 - loss: 0.3761 - val_accuracy: 0.8662 - val_loss: 0.3867
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8786 - loss: 0.3703 - val_accuracy: 0.8662 - val_loss: 0.3787
Epoch 34/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9016 - loss: 0.3376 - val_accuracy: 0.8662 - val_loss: 0.3697
Epoch 35/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8601 - loss: 0.3652 - val_accuracy: 0.8662 - val_loss: 0.3596
Epoch 36/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8949 - loss: 0.3218 - val_accuracy: 0.8732 - val_loss: 0.3467
Epoch 37/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8980 - loss: 0.3076 - val_accuracy: 0.8662 - val_loss: 0.3319
Epoch 38/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8851 - loss: 0.3074 - val_accuracy: 0.8803 - val_loss: 0.3193
Epoch 39/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9126 - loss: 0.2958 - val_accuracy: 0.8873 - val_loss: 0.3081
Epoch 40/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9225 - loss: 0.2706 - val_accuracy: 0.8873 - val_loss: 0.2970
<keras.src.callbacks.history.History at 0x7eca28eddf90>
```

Balanced Results

- Balancing the dataset significantly improves model performance across all metrics.

Optimal Model:

- **Option 11: Linear SVC**
  - It achieves the best F1-score (96%) for fraud detection.
  - Precision (97%) ensures fewer false positives, which is critical in fraud detection.
  - High recall (94%) ensures most fraud cases are caught.

	Class	Precision	Recall	F1-Score	Support
Raw Data (all Not Fraud 100%)					
Option 1: Logistic Regression	Fraud	83%	56%	67%	36
Option 2: Shallow Neural Network	Fraud	75%	75%	75%	36
Option 3: RandomForest	Fraud	77%	47%	59%	36
Option 4: Gradient Boost	Fraud	67%	67%	67%	36
Option 5: Linear SVC	Fraud	7%	97%	14%	36
Balanced					
Option 6: Logistic Regression	Not Fraud	96%	93%	94%	72
	Fraud	93%	96%	94%	70
Option 7: Shallow Neural Network (2 relu)	Not Fraud	89%	100%	94%	72
	Fraud	100%	87%	93%	70
Option 8: Shallow Neural Network (1 relu)	Not Fraud	89%	100%	94%	72
	Fraud	100%	87%	93%	70
Option 9: Random Forest Model	Not Fraud	92%	97%	95%	72
	Fraud	97%	91%	94%	70
Option 10: Gradient Boost	Not Fraud	81%	100%	89%	72
	Fraud	100%	76%	86%	70
Option 11: Linear SVC	Not Fraud	95%	97%	96%	72
	Fraud	97%	94%	96%	70

3 Sentiment and Aspect-Based Analysis on Amazon Reviews

GOALS:

Understand customer feedback trends.

Use insights to suggest actionable improvements.

Highlight product issues (e.g., durability, price).

KEY LIBRARIES:

- PANDAS AND NUMPY: FOR STRUCTURED DATA MANIPULATION.
- NLTK AND TEXTBLOB: FOR SENTIMENT ANALYSIS AND POLARITY SCORING.
- SCIKIT-LEARN: FOR MACHINE LEARNING MODELS LIKE LOGISTIC REGRESSION AND TF-IDF VECTORIZATION.
- MATPLOTLIB AND SEABORN: FOR CREATING DATA VISUALIZATIONS.
- PYLDAVIS: FOR INTERACTIVE VISUALIZATION OF TOPICS.
- TQDM: FOR DISPLAYING PROGRESS BARS DURING HEAVY COMPUTATIONS.
- NLP TOOLS (NLTK, TEXTBLOB) ARE ESSENTIAL FOR EXTRACTING MEANINGFUL INSIGHTS FROM UNSTRUCTURED TEXT.
- HUGGING FACE

What was done ?

Different Approaches

Sentiment Analysis.

Aspect-Based Sentiment Analysis.

TF-IDF and Machine Learning.

Topic Modeling with LDA

Visualizations

Sentiment Analysis

What It Is: Assigning a sentiment score (positive or negative) to each review.

01

I used VADER (from nltk) to calculate compound sentiment scores.

02

- **Categorized sentiments**:
  - Positive: Compound score ≥ 0.05
  - Negative: Compound score < 0.05

03

Why Needed?

- It helps identify overall customer satisfaction levels and common complaints.

Example From Dataset:

Review: "The watch is beautiful but the battery life is terrible."
Sentiment: Negative (compound score: -0.3)

Aspect-Based Sentiment Analysis

What It Is: Analyzing sentiments about specific product aspects (e.g., design, durability).

Approach

- Defined key aspects: design, price, durability, brand.
- Used TextBlob to determine sentiment polarity for each aspect mentioned in a review.

Review: "The design is sleek and modern, but the strap broke in a week."
Design: Positive Durability: Negative

Why Needed?

Pinpoints the areas customers love or dislike, enabling targeted product improvements.

TF-IDF and Machine Learning

What It Is: Convert text data into a numerical format and train a sentiment classifier.

Approach

- Used TfidfVectorizer with n-grams (uni-grams and bi-grams).
- Trained a Logistic Regression model with hyperparameter tuning using GridSearchCV.

Review: "Great watch, but overpriced."
TF-IDF: Captures keywords like "great," "overpriced."
Model Prediction: Negative sentiment.

Why Needed?

Automates sentiment classification for scalability and ensures generalization to new reviews.

Topic Modeling with LDA

Identifying hidden themes within the dataset using Latent Dirichlet Allocation.

Approach

Defined 5 topics based on reviews.

- **Topic 1:** "design, sleek, stylish"
- **Topic 2:** "price, affordable, expensive"

Why Needed?

Summarizes common themes in customer feedback, providing insights into trending issues or preferences.

Star rating - Hugging face vs Regression

Pre-Trained Model: distilbert-base-uncased

- Used AutoTokenizer for truncation and padding (max length: 512).
- Tokenized in chunks for memory efficiency.
- Dataset Class: SentimentDataset for managing tokenized encodings.

Why Hugging Face?

- Leverages Transformers for context-aware analysis.
- Pre-trained embeddings improve accuracy without requiring large datasets.

Training Configuration:

- Epochs: 3
- Batch Size: 8
- Mixed Precision Enabled (fp16) for faster training.

Model Comparison

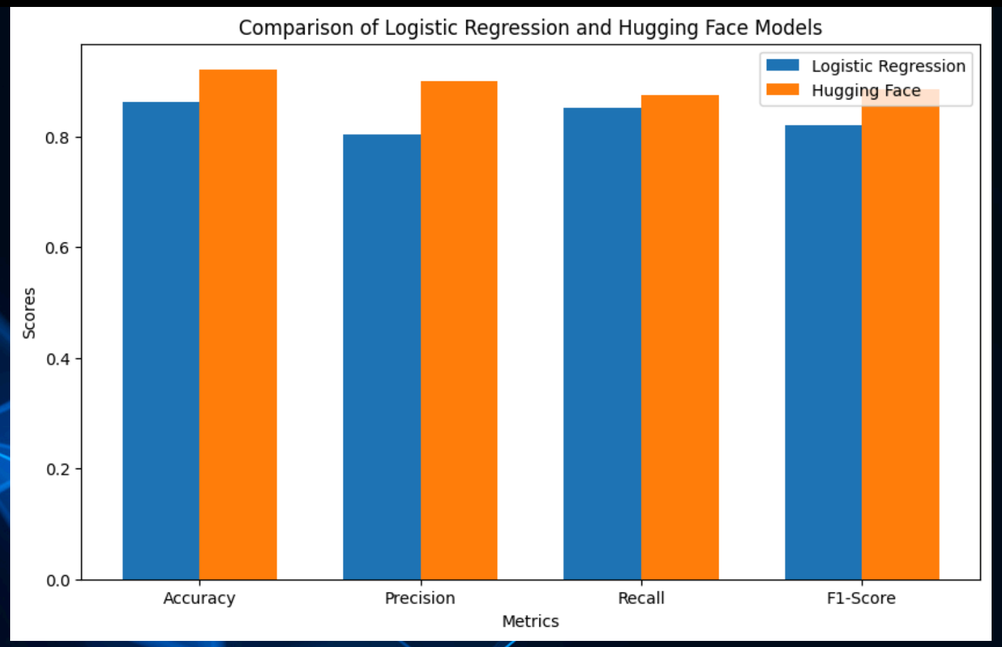
A. Hugging Face Model:

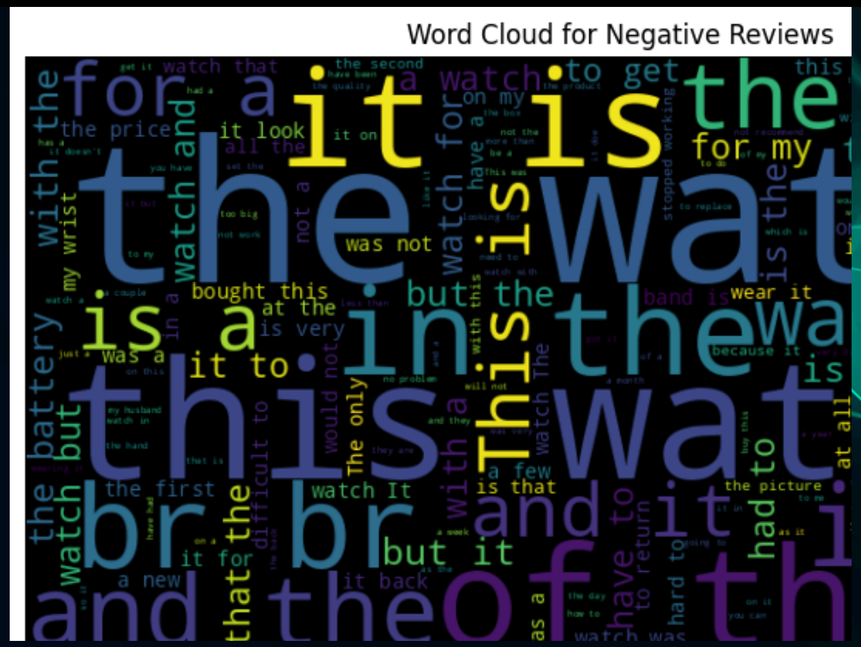
a. Significantly higher F1-scores for both positive and negative classes.
b. Excellent overall accuracy (92%), making it more reliable for predictions.

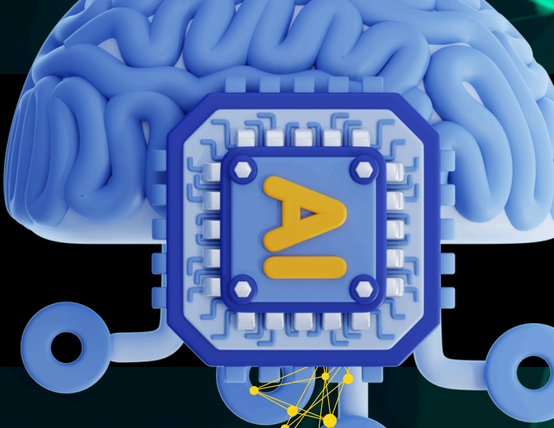
B. Logistic Regression:

a. Struggles with the negative class (low precision of 0.66).
b. Lower accuracy (86%), indicating weaker performance compared to Hugging Face.

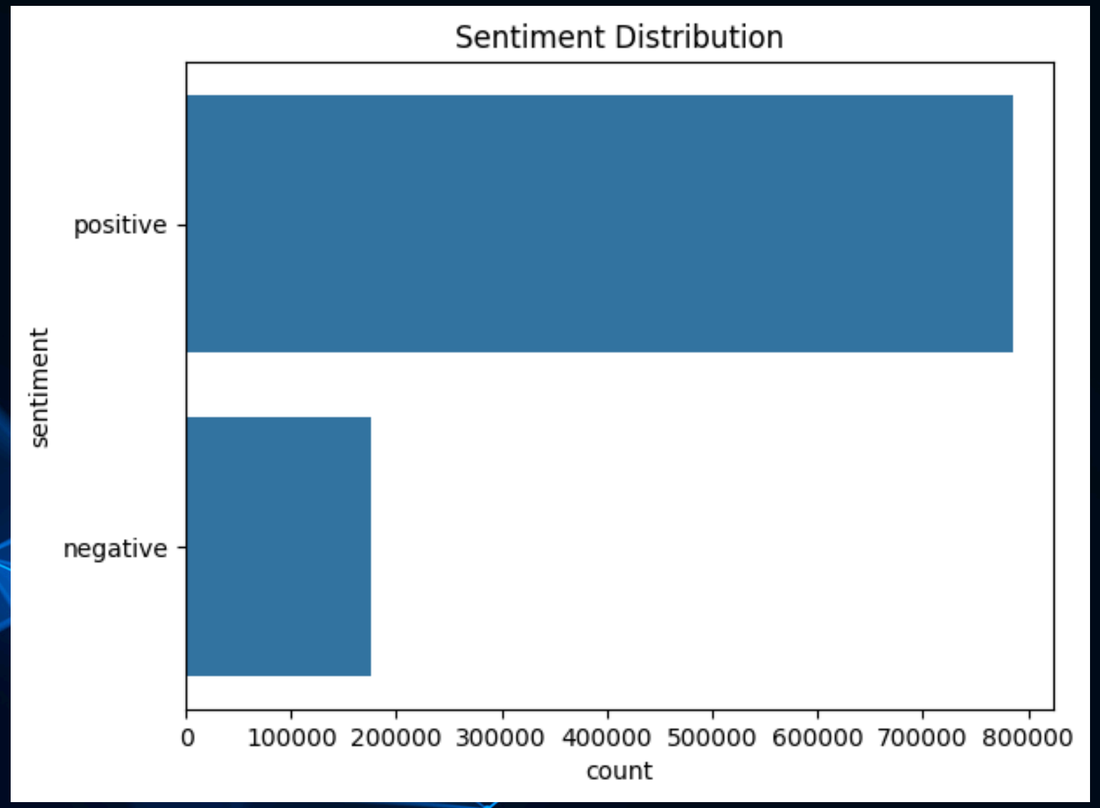
Visualizations

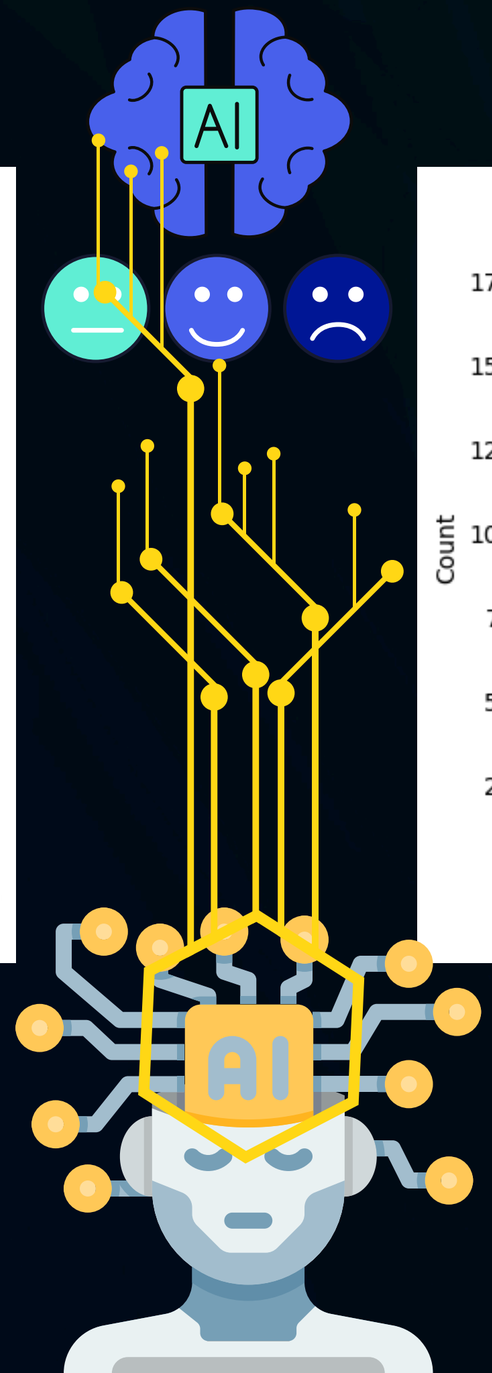


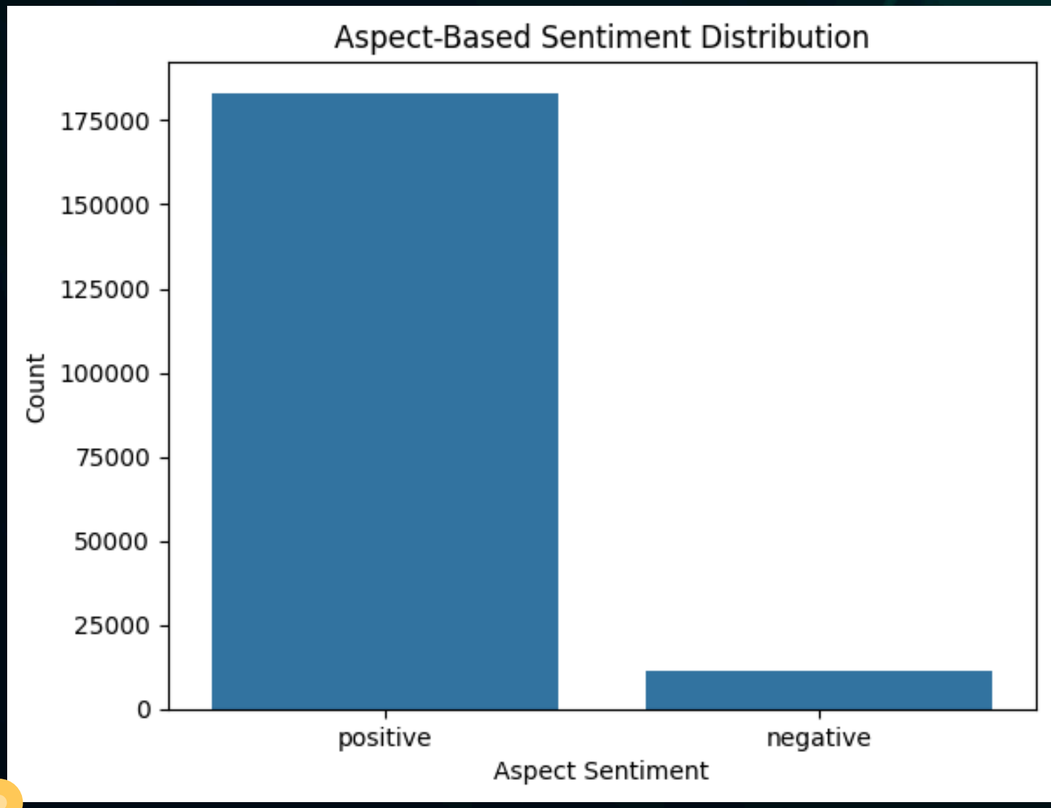




VISUALIZATIONS







NEXT STEPS

Future Work

- Improvements:
  - Experiment with larger Hugging Face models (e.g., BERT, RoBERTa).
  - Extend analysis to multiclass classification (e.g., neutral sentiments).
  - Optimize TF-IDF features for better Logistic Regression performance.
- Applications:
  - Use insights for improving product design based on sentiment trends.
  - Incorporate multilingual reviews for a global analysis.

Model Comparison

THANK YOU!

Q & A

William Aromando, Dan Becker,
Joanne Donohue, Vivin Rajagopalan,
and Chace Snedecor