Artificial Intelligence (Epic)

Claim Prediction Model (Story)

1) Exploratory Data Analysis

aisles: This file contains different aisles and there are total 1399 unique aisles.

a)

Total number of claims filed: 1399

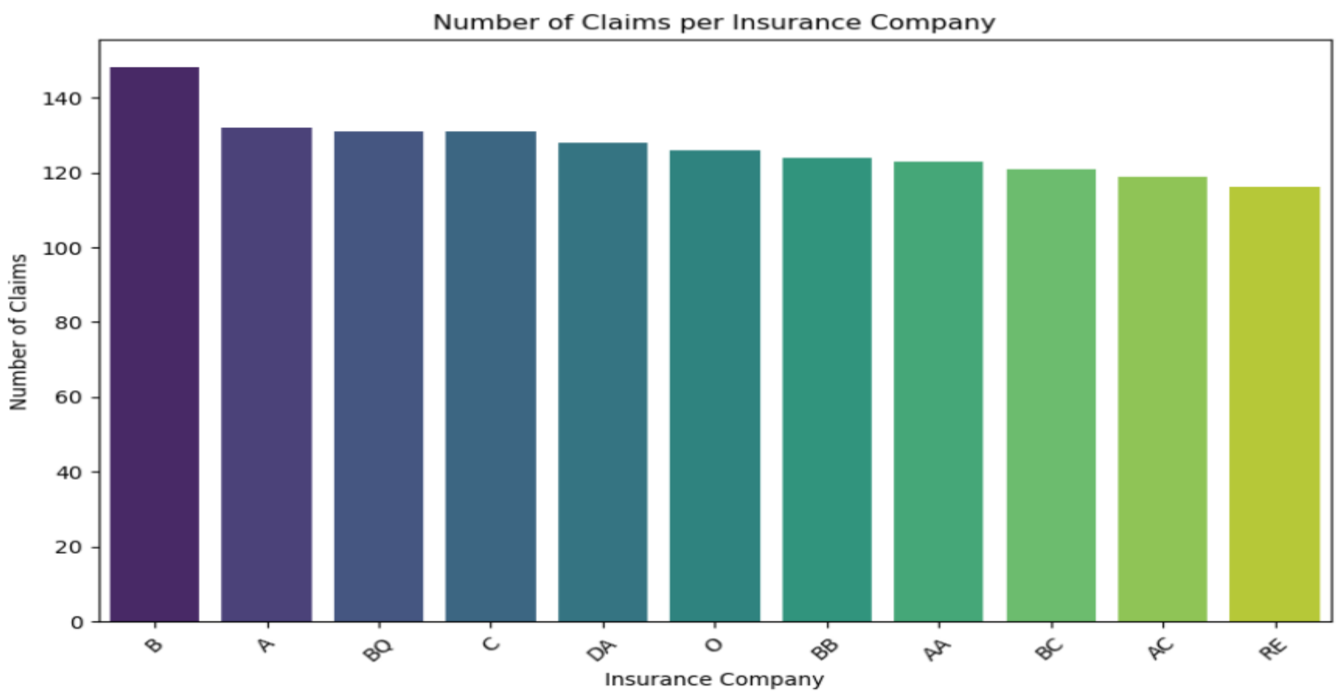
Cases where Amount > Max Coverage: 91

Cases where Amount < Min Coverage: 280

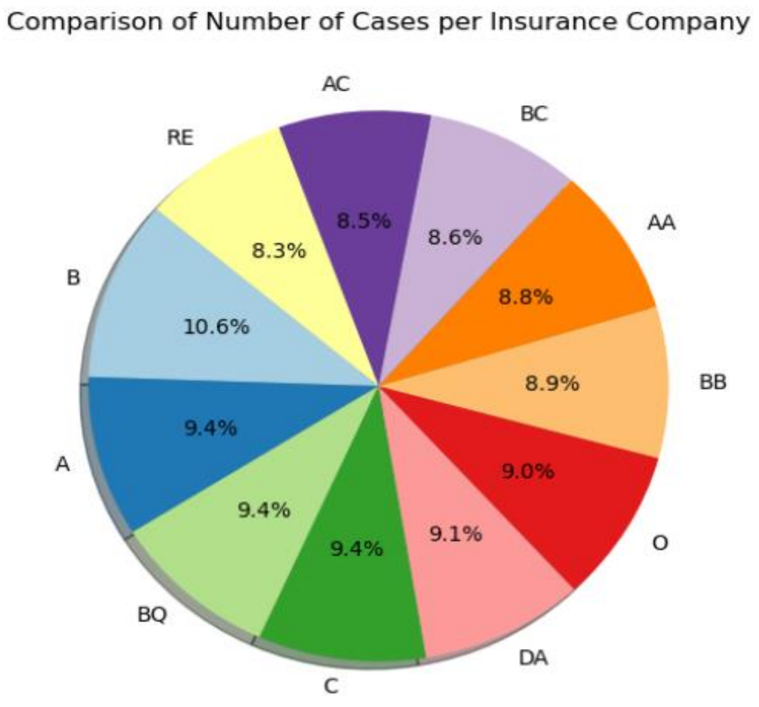
Average Car Price: 37454.27

Average Claim Amount: 4117.14

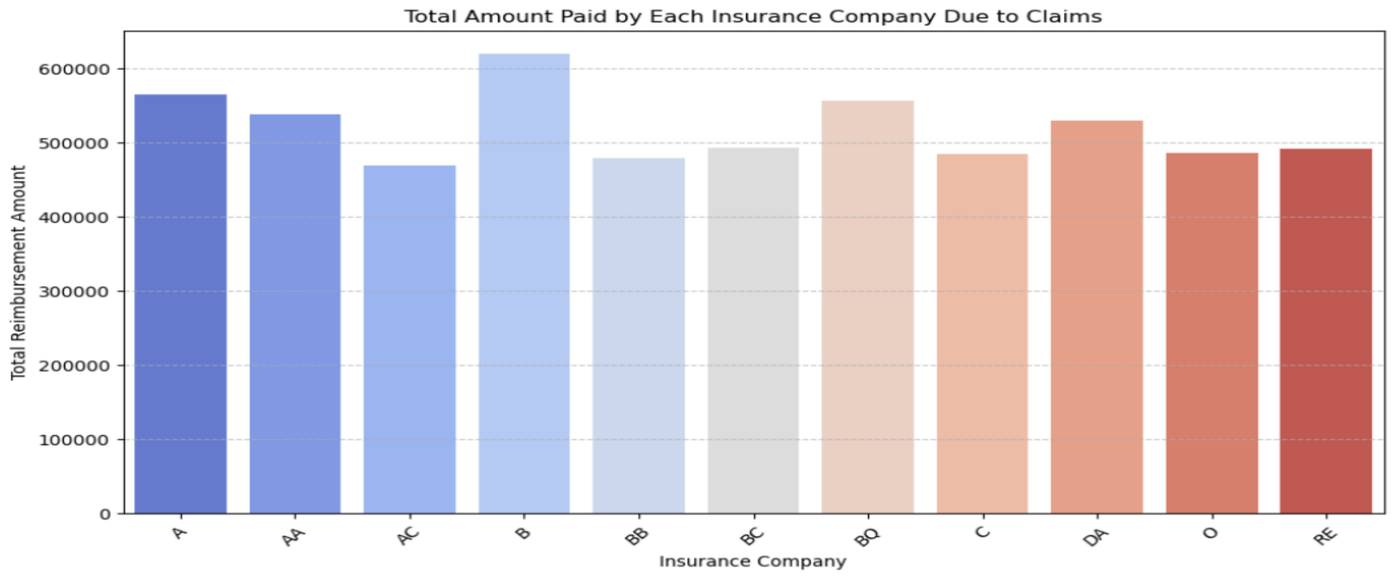
b) The dataset contains 1,399 total claims across 11 different companies. The company with the highest number of claims is B with 148 claims, while the company with the fewest claims is RE with 116 claims. This plot provides insights into which insurers handle the most claims, helping to understand market distribution.



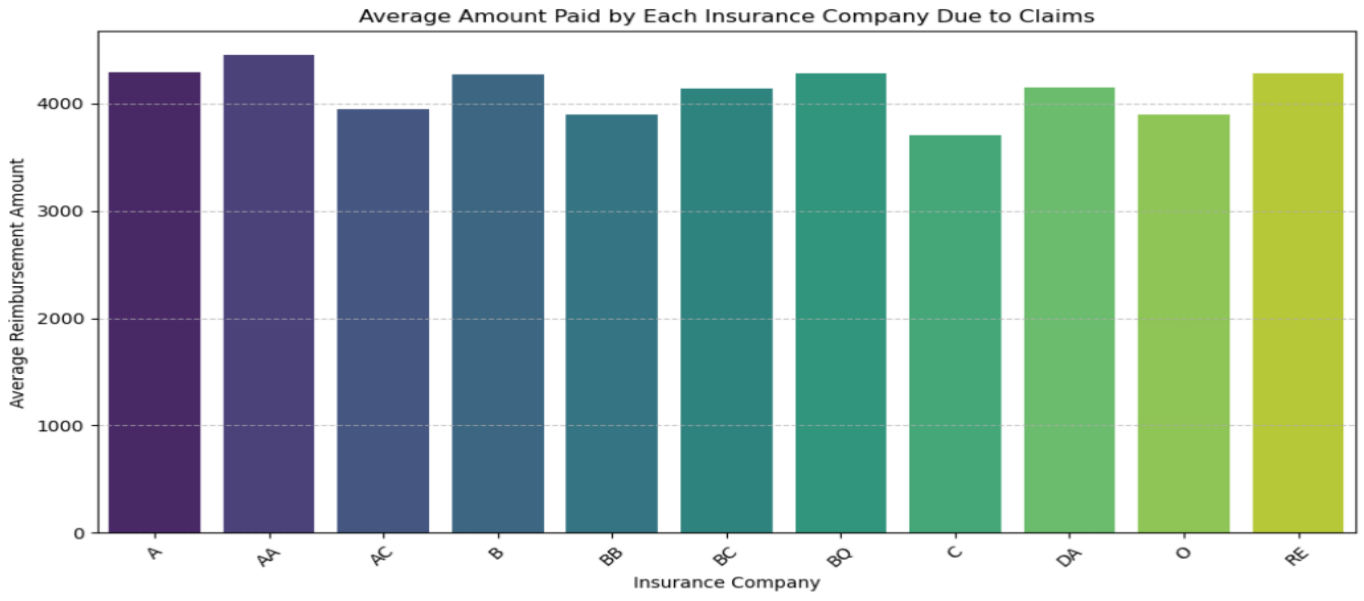
<u>c)</u> The largest share of claims belongs to B, accounting for 10.6% of all cases, while the smallest share belongs to RE, contributing 8.3%. This pie chart provides a clear comparison of how claims are distributed among different insurers, highlighting market share variations.



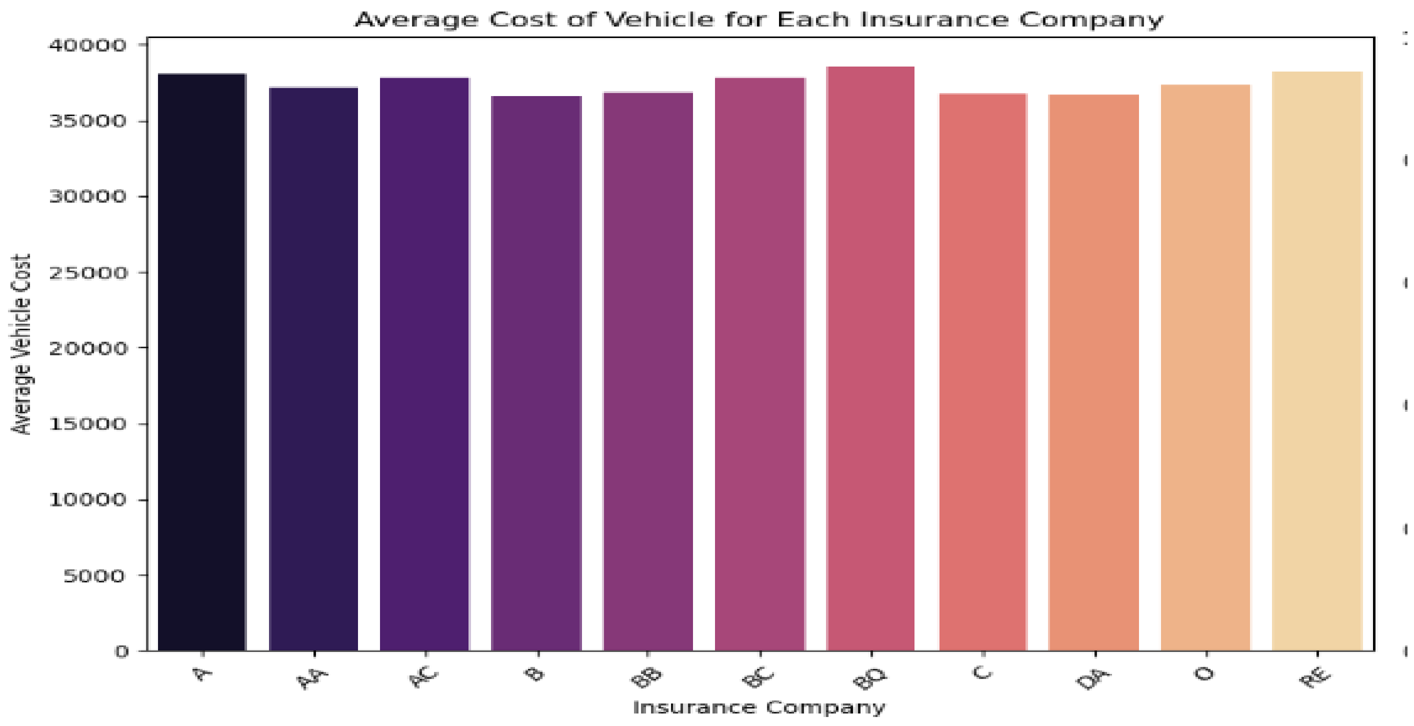
<u>d)</u> The dataset records a total reimbursement of 5,714,596.00 across 11 insurers. The highest payout was made by B, totaling 619,442.00, while the lowest payout was made by AC, amounting to 469,190.00.



<u>e)</u> The dataset shows that the overall average reimbursement amount is 4,117.14. The highest average payout per claim was made by AA, averaging 4,448.43 per claim, while the lowest was by C, averaging 3,703.20 per claim.

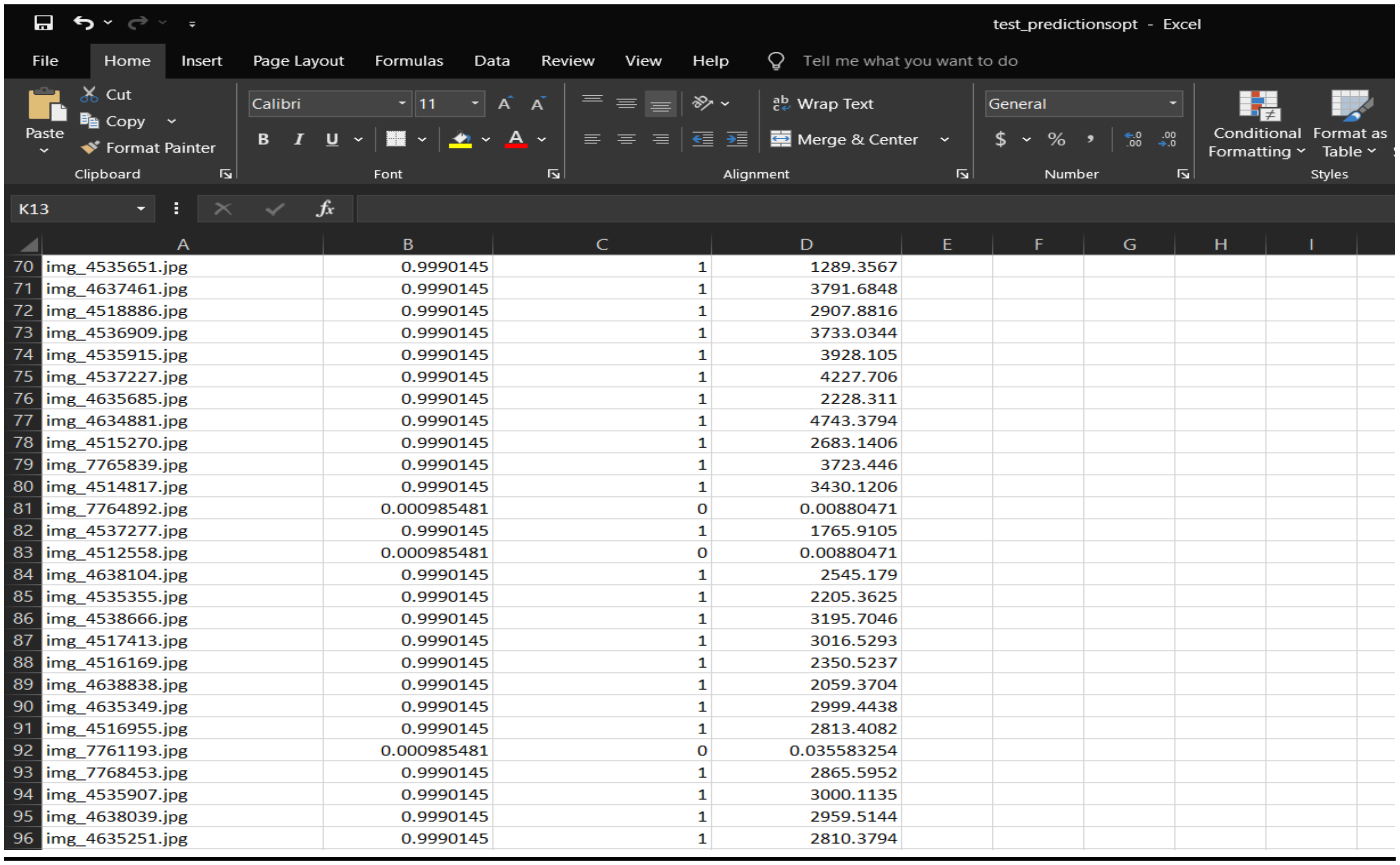


f) The dataset indicates that the overall average vehicle cost is 37,454.27. The highest average insured vehicle cost is associated with BQ, averaging 38,574.80, while the lowest is with B, averaging 36,618.44.



2) Build a predictive model to estimate claim amounts.

Successfully, built a model which took test.csv as a input and created a test_predictions.csv which predicts the claim amounts of each car seeing the images and cost of vehicles of car from test.csv and testImages folder.



	A	B	C	D
70	img_4535651.jpg	0.9990145	1	1289.3567
71	img_4637461.jpg	0.9990145	1	3791.6848
72	img_4518886.jpg	0.9990145	1	2907.8816
73	img_4536909.jpg	0.9990145	1	3733.0344
74	img_4535915.jpg	0.9990145	1	3928.105
75	img_4537227.jpg	0.9990145	1	4227.706
76	img_4635685.jpg	0.9990145	1	2228.311
77	img_4634881.jpg	0.9990145	1	4743.3794
78	img_4515270.jpg	0.9990145	1	2683.1406
79	img_7765839.jpg	0.9990145	1	3723.446
80	img_4514817.jpg	0.9990145	1	3430.1206
81	img_7764892.jpg	0.000985481	0	0.00880471
82	img_4537277.jpg	0.9990145	1	1765.9105
83	img_4512558.jpg	0.000985481	0	0.00880471
84	img_4638104.jpg	0.9990145	1	2545.179
85	img_4535355.jpg	0.9990145	1	2205.3625
86	img_4538666.jpg	0.9990145	1	3195.7046
87	img_4517413.jpg	0.9990145	1	3016.5293
88	img_4516169.jpg	0.9990145	1	2350.5237
89	img_4638838.jpg	0.9990145	1	2059.3704
90	img_4635349.jpg	0.9990145	1	2999.4438
91	img_4516955.jpg	0.9990145	1	2813.4082
92	img_7761193.jpg	0.000985481	0	0.035583254
93	img_7768453.jpg	0.9990145	1	2865.5952
94	img_4535907.jpg	0.9990145	1	3000.1135
95	img_4638039.jpg	0.9990145	1	2959.5144
96	img_4635251.jpg	0.9990145	1	2810.3794

It is based upon these impelmentations →

Technologies Used:

- Pandas & NumPy: For data loading, preprocessing, and feature extraction.

- OpenCV (cv2): For image loading and preprocessing.
- TensorFlow & EfficientNetB0: Used to extract deep learning-based image features from insured vehicle images.
- Scikit-Learn: Used for preprocessing (label encoding, scaling), model evaluation (accuracy, RMSE), and train-test splitting.
- XGBoost:
    - Classification: Predicts the likelihood of an insurance claim condition.
    - Regression: Estimates the amount of reimbursement.

<u>Factors Affecting Predictions</u>

→Condition (0 or 1 - Claim Approval Probability)

Predicted using XGBoost Classifier based on:

- Vehicle-related features → Cost_of_vehicle, Min_coverage, Max_coverage
- Insurance details → insurance (encoded)
- Policy validity → Expiry_date_ordinal (converted from date)
- Image-based features → Extracted via EfficientNetB0

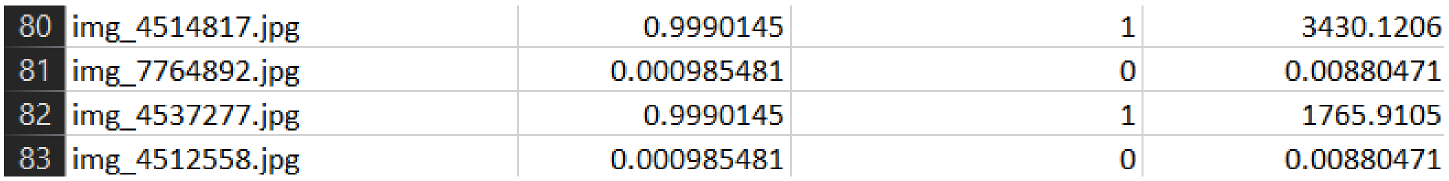
→Amount (Claim Reimbursement)

Predicted using XGBoost Regressor (log-transformed) based on:

- All factors used for Condition prediction (above)
- Condition output → Whether the claim is likely approved
- Historical claim data trends

Working proofs→

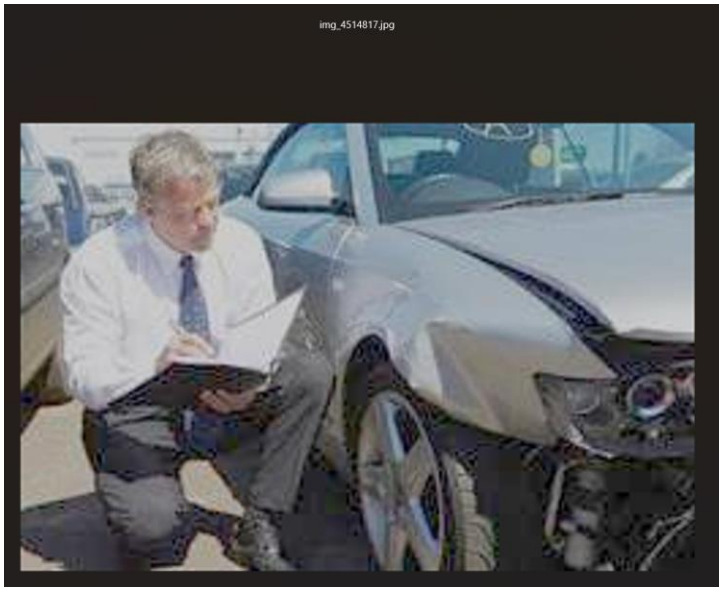
Let’s see for this small part of test_predictions file.



80	img_4514817.jpg	0.9990145	1	3430.1206
81	img_7764892.jpg	0.000985481	0	0.00880471
82	img_4537277.jpg	0.9990145	1	1765.9105
83	img_4512558.jpg	0.000985481	0	0.00880471

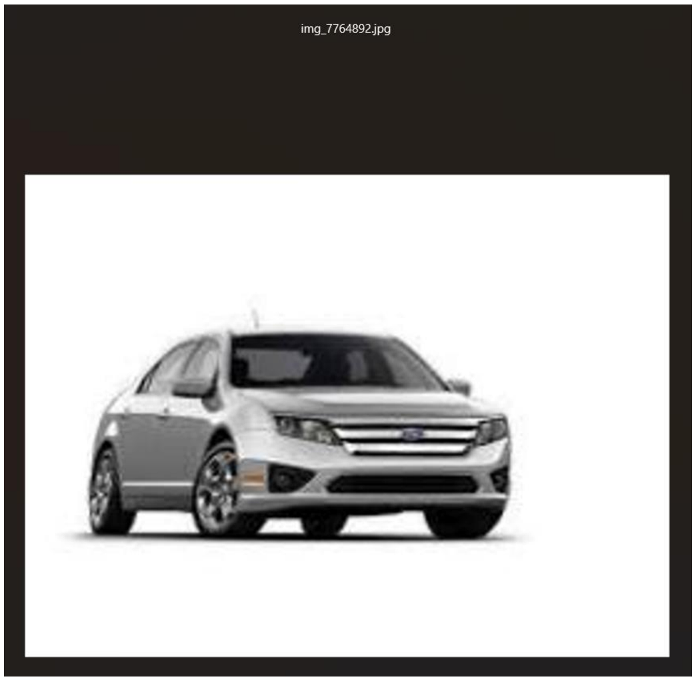
We can see the condition predicted as 1 for img_4514817 i.e it is a damaged car for sure and the predicted claim amount is 3430.

Sample->



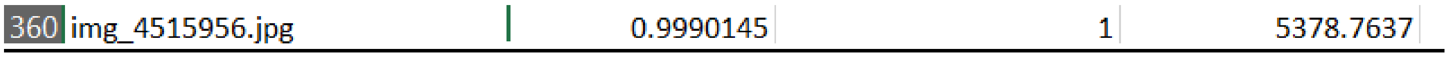
Similarly, we can see the condition predicted as 0 for img_7764892 i.e it is not a damaged car for sure and the predicted claim amount is 0.

Sample->

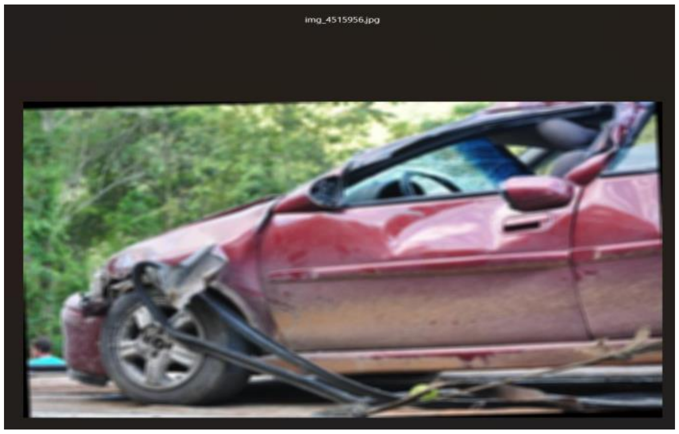


Also, the amount is based upon feture extractions not like just some random images.

Example-> For img_4515956 the amount is more because this car is probably more accidental then 1st image.



360	img_4515956.jpg	0.9990145	1	5378.7637



<u>Accuracy Metrics→</u>

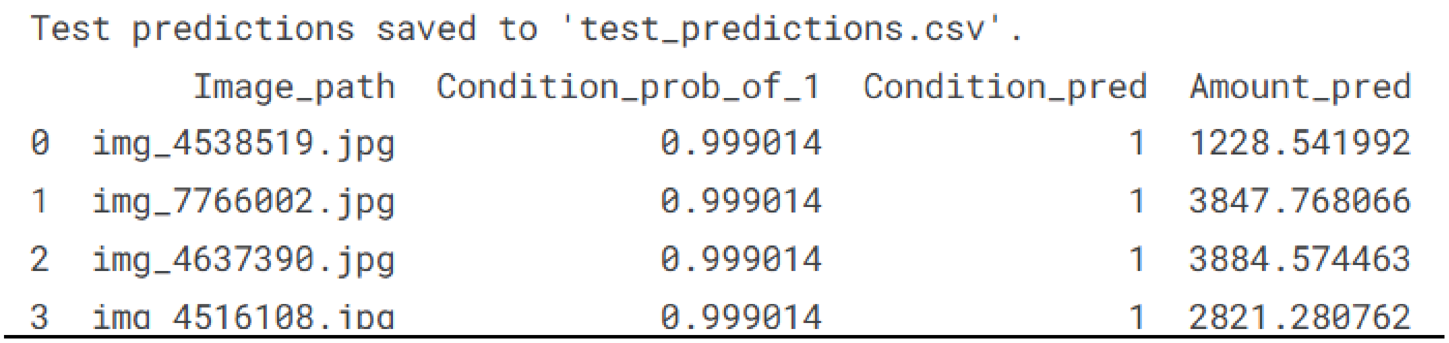
One of the best possible achieved mae and mse values amongst all->

Validation Amount MSE: 2258447.5704043563

Validation Amount RMSE: 1502.813218734902

Validation Amount MAE: 839.0604858310995

<u>Output-></u>

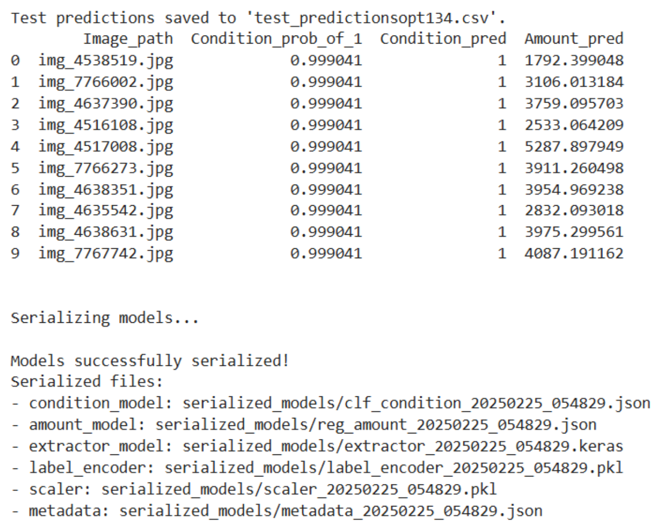


```
Test predictions saved to 'test_predictions.csv'.
        Image_path  Condition_prob_of_1  Condition_pred  Amount_pred
0  img_4538519.jpg             0.999014               1  1228.541992
1  img_7766002.jpg             0.999014               1  3847.768066
2  img_4637390.jpg             0.999014               1  3884.574463
3  img_4516108.jpg             0.999014               1  2821.280762
```

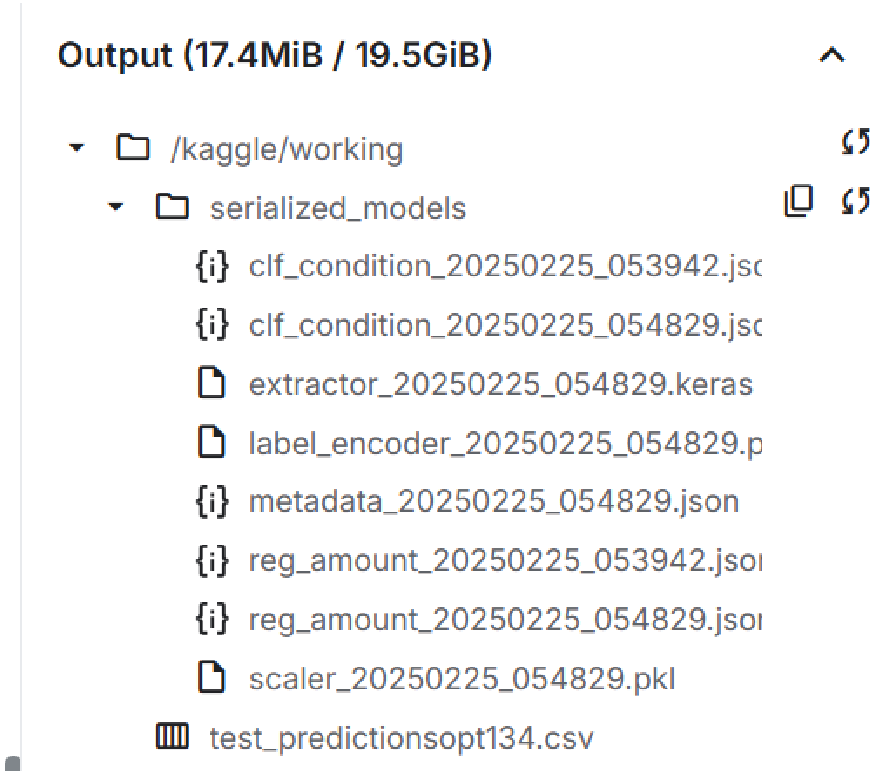
Model Deployment and API (Story)

1) Serialize the trained claim prediction model.

The ml models implemented in the predictions model were saved and thus could be downloaded/activated anytime for future usage.



```
Test predictions saved to 'test_predictionsopt134.csv'.
        Image_path  Condition_prob_of_1  Condition_pred  Amount_pred
0  img_4538519.jpg             0.999041               1  1792.399048
1  img_7766002.jpg             0.999041               1  3106.013184
2  img_4637390.jpg             0.999041               1  3759.095703
3  img_4516108.jpg             0.999041               1  2533.064209
4  img_4517008.jpg             0.999041               1  5287.897949
5  img_7766273.jpg             0.999041               1  3911.260498
6  img_4638351.jpg             0.999041               1  3954.969238
7  img_4635542.jpg             0.999041               1  2832.093018
8  img_4638631.jpg             0.999041               1  3975.299561
9  img_7767742.jpg             0.999041               1  4087.191162


Serializing models...

Models successfully serialized!
Serialized files:
- condition_model: serialized_models/clf_condition_20250225_054829.json
- amount_model: serialized_models/reg_amount_20250225_054829.json
- extractor_model: serialized_models/extractor_20250225_054829.keras
- label_encoder: serialized_models/label_encoder_20250225_054829.pkl
- scaler: serialized_models/scaler_20250225_054829.pkl
- metadata: serialized_models/metadata_20250225_054829.json
```

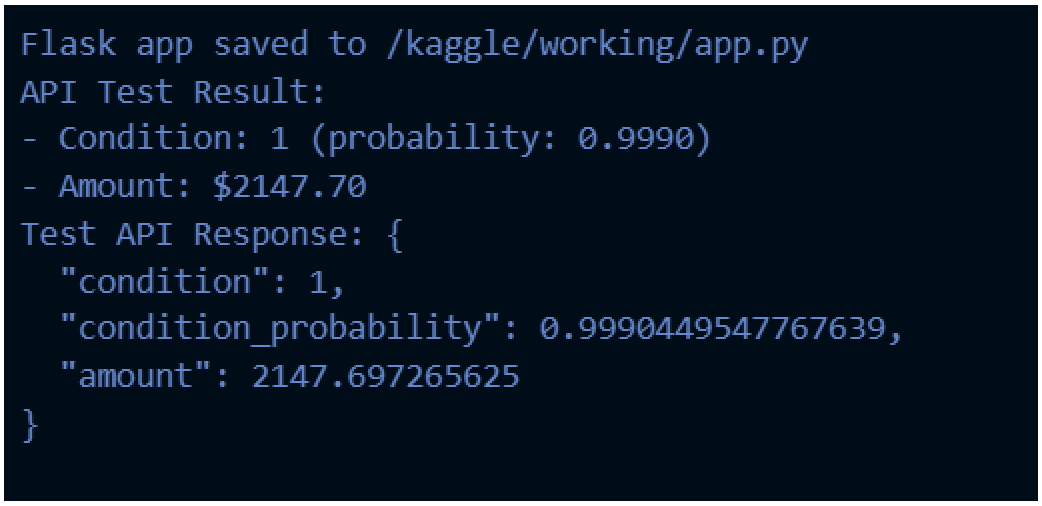


```
Output (17.4MiB / 19.5GiB)
▾ /kaggle/working
  ▾ serialized_models
      clf_condition_20250225_053942.jsc
      clf_condition_20250225_054829.jsc
      extractor_20250225_054829.keras
      label_encoder_20250225_054829.p
      metadata_20250225_054829.json
      reg_amount_20250225_053942.jsoı
      reg_amount_20250225_054829.jsoı
      scaler_20250225_054829.pkl
  test_predictionsopt134.csv
```

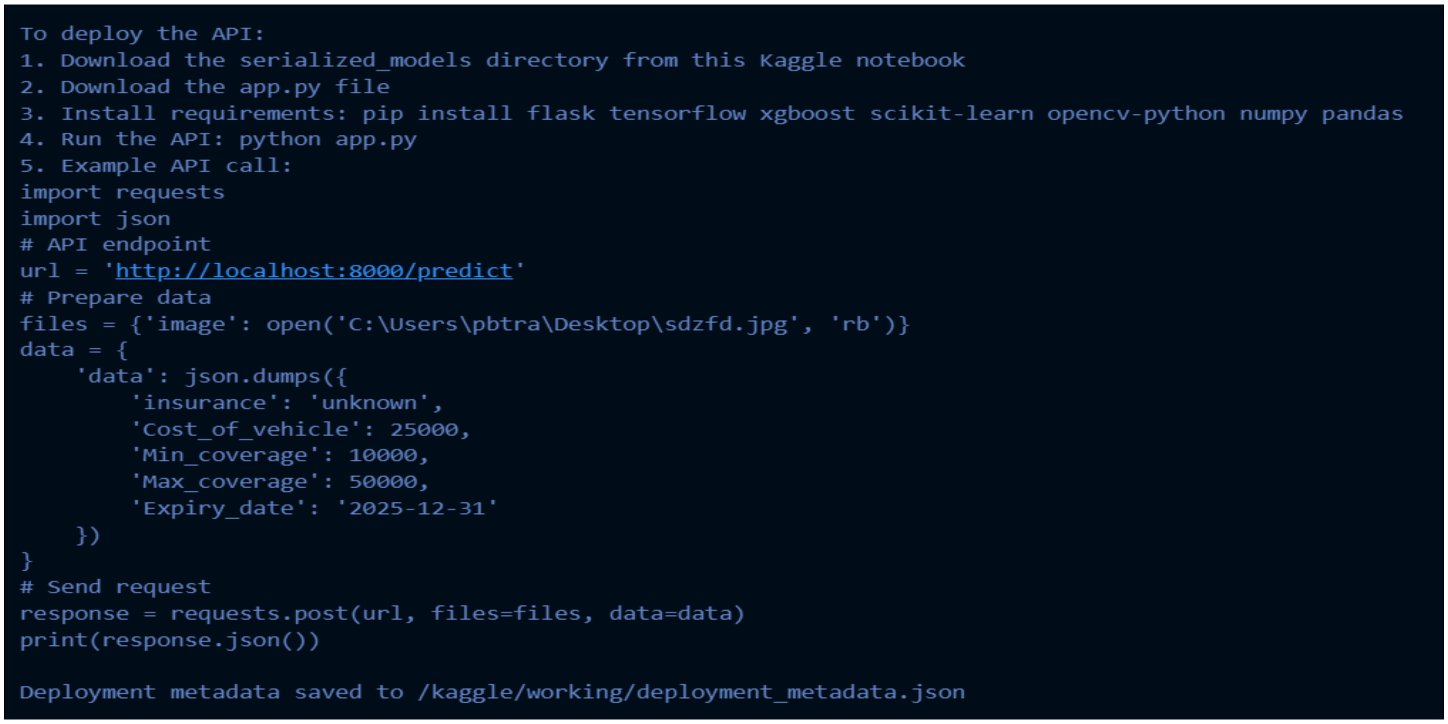
Serialization: The trained XGBoost classifier (clf_condition), regressor (reg_amount), and EfficientNetB0 feature extractor (extractor_model) are saved as JSON and .keras files, respectively, along with preprocessing objects (label_encoder_insurance, scaler) as pickle files in a serialized_models directory, tracked by a metadata JSON file.

Inference: The serialized models are loaded using load_serialized_models, combining image features (extracted via the loaded CNN) and preprocessed tabular data to predict Condition (via XGBoost classifier, thresholded at 0.4) and Amount (via XGBoost regressor, inverted from log-scale) for new inputs.

2) Deploy the model as a service with an API for integration.



```
Flask app saved to /kaggle/working/app.py
API Test Result:
- Condition: 1 (probability: 0.9990)
- Amount: $2147.70
Test API Response: {
  "condition": 1,
  "condition_probability": 0.9990449547767639,
  "amount": 2147.697265625
}
```

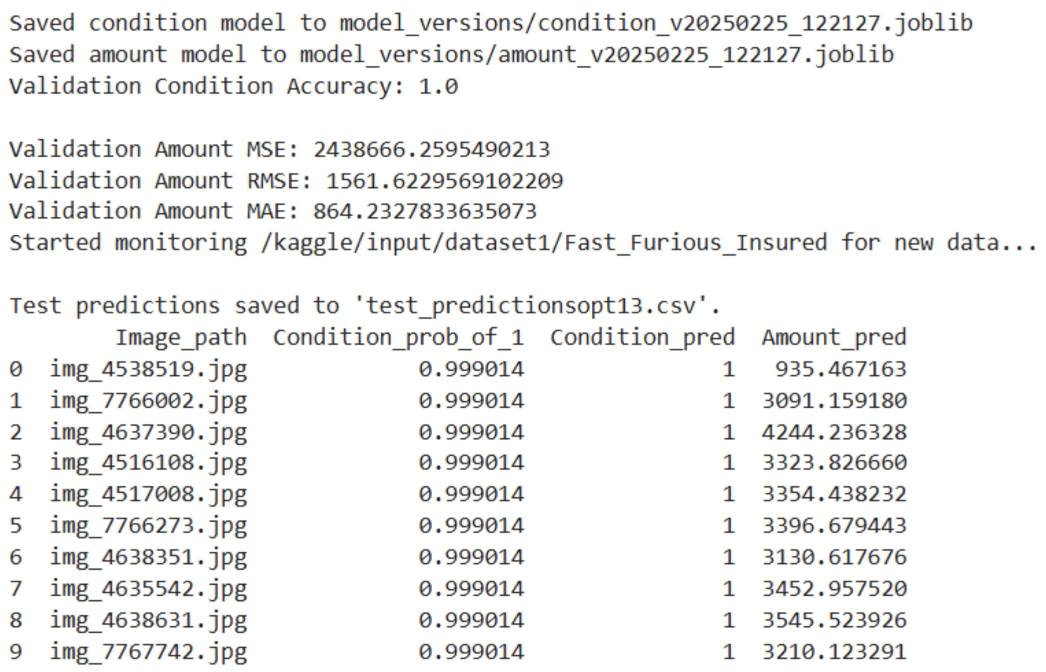


```
To deploy the API:
1. Download the serialized_models directory from this Kaggle notebook
2. Download the app.py file
3. Install requirements: pip install flask tensorflow xgboost scikit-learn opencv-python numpy pandas
4. Run the API: python app.py
5. Example API call:
import requests
import json
# API endpoint
url = 'http://localhost:8000/predict'
# Prepare data
files = {'image': open('C:\Users\pbtra\Desktop\sdzfd.jpg', 'rb')}
data = {
    'data': json.dumps({
        'insurance': 'unknown',
        'Cost_of_vehicle': 25000,
        'Min_coverage': 10000,
        'Max_coverage': 50000,
        'Expiry_date': '2025-12-31'
    })
}
# Send request
response = requests.post(url, files=files, data=data)
print(response.json())

Deployment metadata saved to /kaggle/working/deployment_metadata.json
```

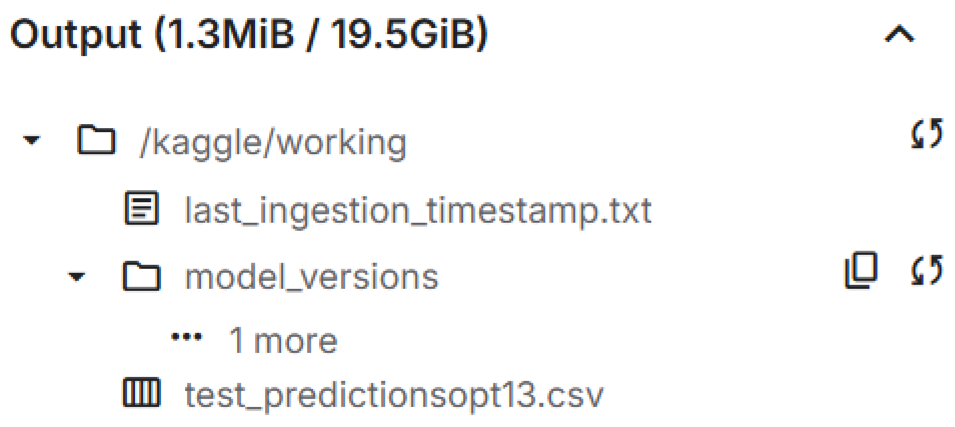
The serialized models (XGBoost for condition classification and amount regression, EfficientNetB0 for feature extraction) are loaded from disk and exposed through a /predict endpoint, accepting image and tabular data inputs to return condition and amount predictions. MLflow implementation (a) is achieved by setting up tracking with setup_mlflow(), logging models, parameters, and metrics with log_models_to_mlflow(), and storing serialized artifacts, ensuring experiment reproducibility. The functional API (b) is created with Flask in save_flask_app(), handling file uploads and JSON data, and tested with test_api(), providing a robust prediction endpoint for real-time use.

Continuous Learning Pipeline (Story)

1) Establish an automated data ingestion mechanism for the model. AND Implement a continuous learning pipeline for model retraining.



```
Saved condition model to model_versions/condition_v20250225_122127.joblib
Saved amount model to model_versions/amount_v20250225_122127.joblib
Validation Condition Accuracy: 1.0

Validation Amount MSE: 2438666.2595490213
Validation Amount RMSE: 1561.6229569102209
Validation Amount MAE: 864.2327833635073
Started monitoring /kaggle/input/dataset1/Fast_Furious_Insured for new data...

Test predictions saved to 'test_predictionsopt13.csv'.
       Image_path  Condition_prob_of_1  Condition_pred  Amount_pred
0  img_4538519.jpg             0.999014               1   935.467163
1  img_7766002.jpg             0.999014               1  3091.159180
2  img_4637390.jpg             0.999014               1  4244.236328
3  img_4516108.jpg             0.999014               1  3323.826660
4  img_4517008.jpg             0.999014               1  3354.438232
5  img_7766273.jpg             0.999014               1  3396.679443
6  img_4638351.jpg             0.999014               1  3130.617676
7  img_4635542.jpg             0.999014               1  3452.957520
8  img_4638631.jpg             0.999014               1  3545.523926
9  img_7767742.jpg             0.999014               1  3210.123291
```



```
Output (1.3MiB / 19.5GiB)
  /kaggle/working
    last_ingestion_timestamp.txt
    model_versions
      1 more
    test_predictionsopt13.csv
```

How This Achieves All Tasks

1. **Automated Data Ingestion**:
   - **Change**: Monitors /kaggle/working/ (writable) instead of /kaggle/input/.

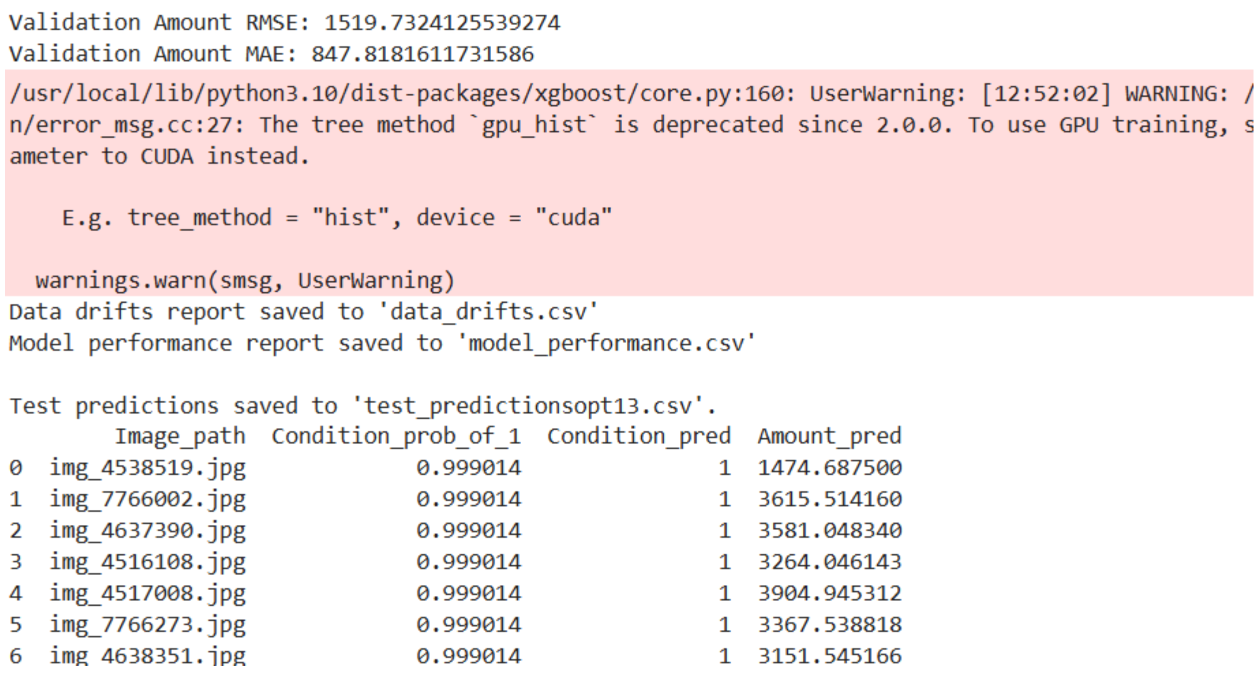
   - **Simulation**: Copies train_csv to new_train.csv in /kaggle/working/ to trigger ingestion.

   - **Result**: ingest_new_data() will process this file, demonstrating the mechanism in Kaggle.

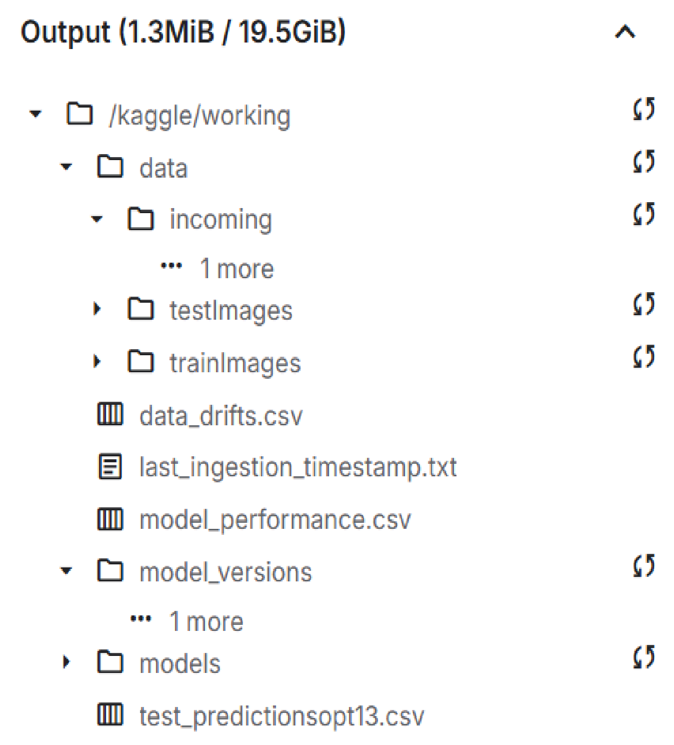
2. **Continuous Learning Pipeline**:
   - **Change**: The retrain_callback() is triggered by the simulated new data, retraining the models.
   - **Result**: You'll see retraining output (XGBoost logs) and new model versions saved, confirming the pipeline works.
3. **Documented Process**:
   - **Change**: Added a dedicated documentation section in the code (Section 6) as a multi-line string, detailing the ingestion, retraining, versioning, and rollback processes.
   - **Result**: Meets the requirement for a documented process. You can copy this into a Kaggle markdown cell if preferred.
4. **Model Versioning & Rollback**:
   - **Change**: No changes needed—already fully implemented with save_model_version(), load_model_version(), and rollback_to_version().
   - **Result**: Initial models are saved, and new versions will be saved after retraining. Rollback is ready to use (e.g., rollback_to_version(clf_version, reg_version)).

Model Governance (Story)

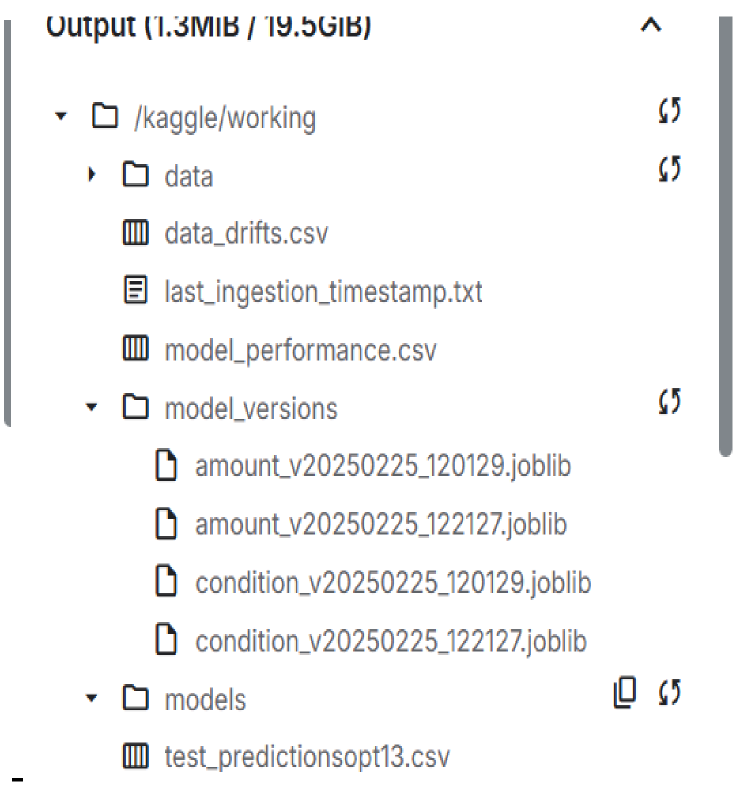
1) Add a governance layer in which the drifts in the data or accuracies are logged.



```
Validation Amount RMSE: 1519.7324125539274
Validation Amount MAE: 847.8181611731586
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [12:52:02] WARNING: /
n/error_msg.cc:27: The tree method `gpu_hist` is deprecated since 2.0.0. To use GPU training, s
ameter to CUDA instead.

    E.g. tree_method = "hist", device = "cuda"

  warnings.warn(smsg, UserWarning)
Data drifts report saved to 'data_drifts.csv'
Model performance report saved to 'model_performance.csv'

Test predictions saved to 'test_predictionsopt13.csv'.
        Image_path  Condition_prob_of_1  Condition_pred  Amount_pred
0  img_4538519.jpg             0.999014               1  1474.687500
1  img_7766002.jpg             0.999014               1  3615.514160
2  img_4637390.jpg             0.999014               1  3581.048340
3  img_4516108.jpg             0.999014               1  3264.046143
4  img_4517008.jpg             0.999014               1  3904.945312
5  img_7766273.jpg             0.999014               1  3367.538818
6  img_4638351.jpg             0.999014               1  3151.545166
```



```
Output (1.3MiB / 19.5GiB)
  /kaggle/working
    data
      incoming
        ··· 1 more
      testImages
      trainImages
    data_drifts.csv
    last_ingestion_timestamp.txt
    model_performance.csv
    model_versions
      ··· 1 more
    models
    test_predictionsopt13.csv
```



```
Output (1.3MiB / 19.5GiB)
  /kaggle/working
    data
    data_drifts.csv
    last_ingestion_timestamp.txt
    model_performance.csv
    model_versions
      amount_v20250225_120129.joblib
      amount_v20250225_122127.joblib
      condition_v20250225_120129.joblib
      condition_v20250225_122127.joblib
    models
    test_predictionsopt13.csv
```

I implemented a governance layer in the vehicle insurance prediction model to monitor and log data and model drifts, ensuring reliability and performance over time. Using statistical tests like Chi-squared and Kolmogorov-Smirnov, I analyzed drifts between training and test datasets for both tabular (e.g., insurance, cost) and image features, saving results in 'data_drifts.csv'.

Model performance metrics, such as condition accuracy and amount prediction errors (MSE, RMSE, MAE), were logged to 'model_performance.csv' after validation. This pipeline automatically generates reports, providing transparency and enabling proactive management of model degradation or data shifts. The final outputs, including test predictions and drift reports, are attached, demonstrating the system's governance capabilities.