Preparation Notebook

Setup Environment

In [1]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
!pip install -q utstd

from utstd.folders import *
from utstd.ipyrenders import *

at = AtFolder(
    course_code=36106,
    assignment="AT1",
)
at.run()

import warnings
warnings.simplefilter(action='ignore')
```

```
[notice] A new release of pip available: 22.3.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
You can now save your data files in: c:\Users\brohao\Desktop\UTS\36106\AT1\36106\assignment\AT
1\data
```

Student Information

In [2]:
```
student_name = "Jiayu Hao"
student_id = "25948860"
```

In [3]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h1", key='student_name', value=student_name)
```

student_name

Jiayu Hao

In [4]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h1", key='student_id', value=student_id)
```

student_id

25948860

---

0. Python Packages

0.a Install Additional Packages

> If you are using additional packages, you need to install them here using the command: `! pip install <package_name>`

In [5]:
```
!pip install numpy
!pip install scikit-learn
```

```
Requirement already satisfied: numpy in c:\users\brohao\appdata\local\programs\python\python31
1\lib\site-packages (2.3.2)

[notice] A new release of pip available: 22.3.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: scikit-learn in c:\users\brohao\appdata\local\programs\python\p
ython311\lib\site-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in c:\users\brohao\appdata\local\programs\python
\python311\lib\site-packages (from scikit-learn) (2.3.2)
Requirement already satisfied: scipy>=1.6.0 in c:\users\brohao\appdata\local\programs\python\p
ython311\lib\site-packages (from scikit-learn) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\brohao\appdata\local\programs\python
\python311\lib\site-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\brohao\appdata\local\programs
\python\python311\lib\site-packages (from scikit-learn) (3.6.0)

[notice] A new release of pip available: 22.3.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

0.b Import Packages

In [6]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
import pandas as pd
import altair as alt
```

In [7]:
```
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

A. Feature Selection

In [8]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
# Load training data
try:
  training_df = pd.read_csv(at.folder_path / "car_insurance_premium_training.csv")
  validation_df = pd.read_csv(at.folder_path / "car_insurance_premium_validation.csv")
  testing_df = pd.read_csv(at.folder_path / "car_insurance_premium_testing.csv")
except Exception as e:
  print(e)
```

A.1 Approach "Drop Personal Infomation"

In [9]:
```
drop_cols = [
    "customer_id","prefix","first_name","last_name","phone_number","email",
    "secondary_address","building_number","street_name","street_suffix","suburb"
]

features_list = [col for col in training_df.columns if col not in drop_cols]
print("Selected features:", features_list)
```

```
Selected features: ['gender', 'birth_date', 'driving_license_date', 'contract_start_date', 'last_renewal_date', 'next_renewal_date', 'distribution_channel', 'seniority', 'current_policies_held', 'max_policies_held', 'max_products_held', 'lapsed_policies', 'lapsed_date', 'payment_method', 'net_premium_amount', 'total_claims_cost_in_current_year', 'total_claims_number_in_current_year', 'total_claims_number_in_history', 'total_claims_number_ratio', 'policy_type', 'second_driver', 'matriculation_year', 'vehicle_horsepower', 'vehicle_cylinder', 'vehicle_value', 'vehicle_doors', 'vehicle_fuel_type', 'vehicle_length', 'vehicle_weight']
```

In [10]:
```python
# provide an explanation on why you use this approach for feature selection and describe its i
feature_selection_1_insights = """
Identifiers and personal information (e.g., customer_id, name, phone_number) do not contribute
High-cardinality categorical features (e.g., street_name, email) lead to sparse data with litt
"""
```

In [11]:
```python
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_selection_1_insights', value=feature_selection_1_insights)
```

feature_selection_1_insights

Identifiers and personal information (e.g., customer_id, name, phone_number) do not contribute to premium prediction and create privacy risks. High-cardinality categorical features (e.g., street_name, email) lead to sparse data with little business value.

A.2 Approach "Delete the features used for derived features"

In [12]:
```python
drop_cols = [
    "contract_start_date","last_renewal_date","next_renewal_date",
    "birth_date","driving_license_date","matriculation_year"
]

features_list = [col for col in features_list if col not in drop_cols]
print("Selected features:", features_list)
```

```
Selected features: ['gender', 'distribution_channel', 'seniority', 'current_policies_held', 'max_policies_held', 'max_products_held', 'lapsed_policies', 'lapsed_date', 'payment_method', 'net_premium_amount', 'total_claims_cost_in_current_year', 'total_claims_number_in_current_year', 'total_claims_number_in_history', 'total_claims_number_ratio', 'policy_type', 'second_driver', 'vehicle_horsepower', 'vehicle_cylinder', 'vehicle_value', 'vehicle_doors', 'vehicle_fuel_type', 'vehicle_length', 'vehicle_weight']
```

In [13]:
```python
# provide an explanation on why you use this approach for feature selection and describe its i
feature_selection_2_insights = """
Original variables used only to create derived features (e.g., birth_date, driving_license_dat
It would duplicate information already captured in derived features (age_at_contract, driving_
"""
```

In [14]:
```python
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_selection_2_insights', value=feature_selection_2_insights)
```

feature_selection_2_insights

Original variables used only to create derived features (e.g., birth_date, driving_license_date, matriculation_year) do not directly improve prediction. It would duplicate information already captured in derived features (age_at_contract, driving_experience, and car_age).

A.3 Approach "Drop features with too many missing values"

> You can add more cells related to other approaches in this section

In [15]:
```
drop_cols = [
    "lapsed_date", "prefix"
]

features_list = [col for col in features_list if col not in drop_cols]
print("Selected features:", features_list)
```

```
Selected features: ['gender', 'distribution_channel', 'seniority', 'current_policies_held', 'm
ax_policies_held', 'max_products_held', 'lapsed_policies', 'payment_method', 'net_premium_amou
nt', 'total_claims_cost_in_current_year', 'total_claims_number_in_current_year', 'total_claims
_number_in_history', 'total_claims_number_ratio', 'policy_type', 'second_driver', 'vehicle_hor
sepower', 'vehicle_cylinder', 'vehicle_value', 'vehicle_doors', 'vehicle_fuel_type', 'vehicle_
length', 'vehicle_weight']
```

In [16]:
```
feature_selection_3_insights = """
Too many missing values exist in lapsed_date(61%), prefix(34%), low predictive value.
"""
```

In [17]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_selection_3_insights', value=feature_selection_3_insights)
```

feature_selection_3_insights

Too many missing values exist in lapsed_date(61%), prefix(34%), low predictive value.

A.4 Final Selection of Features

In [18]:
```
target_name = "net_premium_amount"        # Target variable
#features_list.append(target_name)
print("Selected features:", features_list)
```

```
Selected features: ['gender', 'distribution_channel', 'seniority', 'current_policies_held', 'm
ax_policies_held', 'max_products_held', 'lapsed_policies', 'payment_method', 'net_premium_amou
nt', 'total_claims_cost_in_current_year', 'total_claims_number_in_current_year', 'total_claims
_number_in_history', 'total_claims_number_ratio', 'policy_type', 'second_driver', 'vehicle_hor
sepower', 'vehicle_cylinder', 'vehicle_value', 'vehicle_doors', 'vehicle_fuel_type', 'vehicle_
length', 'vehicle_weight']
```

In [19]:
```
# provide a quick explanation on the features selected
feature_selection_explanations = """
The selected features exclude identifiers and personal information (e.g., customer_id, name, 
high-cardinality categorical features (e.g., street_name, email) that add sparsity without va
and original variables only used to build derived features (e.g., birth_date, driving_license_
Features with excessive missing values and low predictive power (e.g., lapsed_date, prefix) we
"""
```

In [20]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_selection_explanations', value=feature_selection_explanati
```

feature_selection_explanations

The selected features exclude identifiers and personal information (e.g., customer_id, name, phone_number) to avoid privacy risks, high-cardinality categorical features (e.g., street_name, email) that add sparsity without value, and original variables only used to build derived features (e.g., birth_date, driving_license_date, matriculation_year) to prevent duplication. Features with excessive missing values and low predictive power (e.g., lapsed_date, prefix) were also removed.

B. Data Cleaning

In [21]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
# Create copy of datasets
try:
  training_df_clean = training_df[features_list].copy()
  validation_df_clean = validation_df[features_list].copy()
  testing_df_clean = testing_df[features_list].copy()
except Exception as e:
  print(e)
```

B.1 Fixing "Missing Value"

In [22]:
```
def clean_dataset_missing_value(df):

    # Handle missing categorical
    if "vehicle_fuel_type" in df.columns:
        df["vehicle_fuel_type"].fillna("Missing", inplace=True)

    # Handle missing numerical
    if "vehicle_length" in df.columns:
        df["vehicle_length"].fillna(df["vehicle_length"].median(), inplace=True)
```

In [23]:
```
clean_dataset_missing_value(training_df_clean)
clean_dataset_missing_value(validation_df_clean)
clean_dataset_missing_value(testing_df_clean)
```

In [24]:
```
# Provide some explanations on why you believe it is important to fix this issue and its impac
data_cleaning_1_explanations = """
Fixing missing values properly can keep the dataset complete and reliable.
Filling categorical gaps with “Missing” avoids data loss.
Replacing numerical gaps with the median reduces bias.
"""
```

In [25]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_cleaning_1_explanations', value=data_cleaning_1_explanations)
```

data_cleaning_1_explanations

Fixing missing values properly can keep the dataset complete and reliable. Filling categorical gaps with “Missing” avoids data loss. Replacing numerical gaps with the median reduces bias.

B.2 Fixing "Anomalies"

In [26]:
```
def clean_dataset_anomalies(df):
    # Fix vehicle_doors anomalies
    if "vehicle_doors" in df.columns:
        df.loc[df["vehicle_doors"] <= 0, "vehicle_doors"] = 4
```

In [27]:
```
clean_dataset_anomalies(training_df_clean)
clean_dataset_anomalies(validation_df_clean)
clean_dataset_anomalies(testing_df_clean)
```

In [28]:
```
# Provide some explanations on why you believe it is important to fix this issue and its impa
data_cleaning_2_explanations = """
Fixing anomalies can improve data quality (and model accuracy).
Correcting invalid values like zero doors ensures realistic inputs and prevents misleading ef
"""
```

In [29]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_cleaning_2_explanations', value=data_cleaning_2_explanations)
```

data_cleaning_2_explanations

Fixing anomalies can improve data quality (and model accuracy). Correcting invalid values like zero doors ensures realistic inputs and prevents misleading effects on predictions.

B.3 Fixing "Outliers"

In [30]:
```
def clean_dataset_outliers(df):
    # Cap extreme outliers for vehicle_value
    if "vehicle_value" in df.columns:
        df["vehicle_value"] = np.clip(df["vehicle_value"], 500, 100000)
```

In [31]:
```
clean_dataset_outliers(training_df_clean)
clean_dataset_outliers(validation_df_clean)
clean_dataset_outliers(testing_df_clean)
```

In [32]:
```
# Provide some explanations on why you believe it is important to fix this issue and its impa
data_cleaning_3_explanations = """
Fixing outliers can reduce noise and keep predictions stable.
Capping extreme vehicle values prevents tail records from distorting the model.
"""
```

In [33]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_cleaning_3_explanations', value=data_cleaning_3_explanations)
```

data_cleaning_3_explanations

Fixing outliers can reduce noise and keep predictions stable. Capping extreme vehicle values prevents tail records from distorting the model.

C. Feature Engineering

In [34]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
# Create copy of datasets

try:
```

```
    training_df_eng = training_df_clean.copy()
    validation_df_eng = validation_df_clean.copy()
    testing_df_eng = testing_df_clean.copy()
except Exception as e:
    print(e)
```

C.1 New Feature "Age At Contract"

In [35]:
```
def feature_engineering_age(df, df_eng):
    birth = pd.to_datetime(df["birth_date"], errors="coerce", infer_datetime_format=True)
    contract_start = pd.to_datetime(df["contract_start_date"], errors="coerce", infer_datetim

    year_diff = (contract_start.dt.year - birth.dt.year).astype("float")

    before_birthday = (
    (contract_start.dt.month < birth.dt.month) |
    ((contract_start.dt.month == birth.dt.month) & (contract_start.dt.day < birth.dt.day))
    )

    df_eng["age_at_contract"] = (year_diff - before_birthday.astype(int)).where(birth.notna()
```

In [36]:
```
feature_engineering_age(training_df, training_df_eng)
feature_engineering_age(validation_df, validation_df_eng)
feature_engineering_age(testing_df, testing_df_eng)
```

In [37]:
```
# Provide some explanations on why you believe it is important to create this feature and its
feature_engineering_1_explanations = """
It is important to create this feature because customer age at contract start reflects driving
Younger or very old drivers may have higher accident risk, which impacts premium prediction.
"""
```

In [38]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_engineering_1_explanations', value=feature_engineering_1_e
```

feature_engineering_1_explanations

It is important to create this feature because customer age at contract start reflects driving maturity and risk. Younger or very old drivers may have higher accident risk, which impacts premium prediction.

C.2 New Feature "Driving Experience"

In [39]:
```
def feature_engineering_driving_experience(df, df_eng):
    lic = pd.to_datetime(df["driving_license_date"], errors="coerce")
    ref = pd.to_datetime(df["contract_start_date"], errors="coerce")

    year_diff = ref.dt.year - lic.dt.year
    before_anniv = (
        (ref.dt.month < lic.dt.month) |
        ((ref.dt.month == lic.dt.month) & (ref.dt.day < lic.dt.day))
    )
    df_eng["driving_experience"] = (year_diff - before_anniv.astype(int)).where(lic.notna() &
```

In [40]:
```
feature_engineering_driving_experience(training_df, training_df_eng)
feature_engineering_driving_experience(validation_df, validation_df_eng)
feature_engineering_driving_experience(testing_df, testing_df_eng)
```

In [41]:
```
# Provide some explanations on why you believe it is important to create this feature and its
feature_engineering_2_explanations = """
```

```
It is important to create this feature because years of driving experience show how skilled ar
More experience usually lowers accident probability.
"""
```

In [42]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_engineering_2_explanations', value=feature_engineering_2_e:
```

feature_engineering_2_explanations

It is important to create this feature because years of driving experience show how skilled and safe a driver may be. More experience usually lowers accident probability.

C.3 New Feature "Car Age"

In [43]:
```
def feature_engineering_car_age(df, df_eng):
    year = pd.to_numeric(training_df["matriculation_year"], errors="coerce")
    ref  = pd.to_datetime(training_df["contract_start_date"], errors="coerce")

    df_eng["car_age"] = (ref.dt.year - year).where(year.notna() & ref.notna(), np.nan)
```

In [44]:
```
feature_engineering_car_age(training_df, training_df_eng)
feature_engineering_car_age(validation_df, validation_df_eng)
feature_engineering_car_age(testing_df, testing_df_eng)
```

In [45]:
```
# Provide some explanations on why you believe it is important to create this feature and its
feature_engineering_3_explanations = """
It is important to create this feature because vehicle age influences claim cost and risk leve
Older cars may have lower market value but higher breakdown risk.
"""
```

In [46]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='feature_engineering_3_explanations', value=feature_engineering_3_e:
```

feature_engineering_3_explanations

It is important to create this feature because vehicle age influences claim cost and risk level. Older cars may have lower market value but higher breakdown risk.

D. Data Preparation for Modeling

In [47]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
# Create copy of datasets

try:
  X_train = training_df_eng.copy()
  X_val = validation_df_eng.copy()
  X_test = testing_df_eng.copy()

  y_train = X_train.pop(target_name)
  y_val = X_val.pop(target_name)
  y_test = X_test.pop(target_name)
except Exception as e:
  print(e)
```

D.1 Data Transformation `log1p`

In [48]:
```
num_features = ["seniority","current_policies_held","max_policies_held","lapsed_policies",
                "total_claims_number_in_history","total_claims_number_in_current_year",
                "total_claims_cost_in_current_year","total_claims_number_ratio",
                "vehicle_value","vehicle_horsepower",
                "vehicle_cylinder","vehicle_weight","vehicle_length",
                "age_at_contract","driving_experience","car_age"]
log1p_cols = [
    "vehicle_value",
    "total_claims_cost_in_current_year",
    "total_claims_number_in_current_year",
    "total_claims_number_in_history",
    "total_claims_number_ratio",
]
```

In [49]:
```
# Strongly right-skewed columns to log1p (adjust as needed)
for c in [c for c in log1p_cols if c in num_features]:
    for X_ in (X_train, X_val, X_test):
        pos = X_[c] > 0
        X_.loc[pos, c] = np.log1p(X_.loc[pos, c])
```

In [50]:
```
# Provide some explanations on why you believe it is important to perform this data transforma
data_transformation_1_explanations = """
Log1p reduces skewness in highly skewed features.
This makes the data more balanced, improves model stability, and helps predictions become more
"""
```

In [51]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_transformation_1_explanations', value=data_transformation_1_e
```

data_transformation_1_explanations

Log1p reduces skewness in highly skewed features. This makes the data more balanced, improves model stability, and helps predictions become more accurate.

D.2 Data Transformation Scaling

In [52]:
```
num_features = ["seniority","current_policies_held","max_policies_held","lapsed_policies",
                "total_claims_number_in_history","total_claims_number_in_current_year",
                "total_claims_cost_in_current_year","total_claims_number_ratio",
                "vehicle_value","vehicle_horsepower",
                "vehicle_cylinder","vehicle_weight","vehicle_length",
                "age_at_contract","driving_experience","car_age"]
```

In [53]:
```
num_scaler = StandardScaler()
X_train[num_features] = num_scaler.fit_transform(X_train[num_features])
X_val[num_features]   = num_scaler.transform(X_val[num_features])
X_test[num_features]  = num_scaler.transform(X_test[num_features])

print("Numeric transformed (in-place). Shapes:",
      X_train[num_features].shape, X_val[num_features].shape, X_test[num_features].shape)
```

```
Numeric transformed (in-place). Shapes: (32136, 16) (10700, 16) (10666, 16)
```

In [54]:
```
# Provide some explanations on why you believe it is important to perform this data transforma
data_transformation_2_explanations = """
Scaling puts all numerical features on the same scale.
```

```
This prevents large-value features from dominating distance-based models.
"""
```

In [55]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_transformation_2_explanations', value=data_transformation_2_e
```

data_transformation_2_explanations

Scaling puts all numerical features on the same scale. This prevents large-value features from dominating distance-based models.

D.3 Data Transformation One-hot encoding

In [56]:
```
cat_features = ["gender","policy_type","second_driver","payment_method",
                "vehicle_fuel_type","distribution_channel"]
bad_mask_train = X_train["distribution_channel"] == "00/01/1900"
bad_mask_val   = X_val["distribution_channel"] == "00/01/1900"
bad_mask_test  = X_test["distribution_channel"] == "00/01/1900"

print("Bad rows in train:", bad_mask_train.sum())
print("Bad rows in val:", bad_mask_val.sum())
print("Bad rows in test:", bad_mask_test.sum())

bad_value = "00/01/1900"

for df in (X_train, X_val, X_test):
    df["distribution_channel"] = df["distribution_channel"].apply(
        lambda v: "Missing" if v == bad_value else v
    )

print("Unique distribution_channel after cleaning:")
print(X_train["distribution_channel"].value_counts())
```

```
Bad rows in train: 939
Bad rows in val: 372
Bad rows in test: 308
Unique distribution_channel after cleaning:
distribution_channel
0          15662
1          15535
Missing      939
Name: count, dtype: int64
```

In [57]:
```
ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
ohe_cols = ohe.fit(X_train[cat_features]).get_feature_names_out(cat_features)

X_train[ohe_cols] = ohe.transform(X_train[cat_features])
X_val[ohe_cols]   = ohe.transform(X_val[cat_features])
X_test[ohe_cols]  = ohe.transform(X_test[cat_features])

X_train.drop(columns=cat_features, inplace=True)
X_val.drop(columns=cat_features, inplace=True)
X_test.drop(columns=cat_features, inplace=True)

print("Categorical transformed (in-place). New OHE cols:", len(ohe_cols))
print("Final shapes:", X_train.shape, X_val.shape, X_test.shape)
```

```
Categorical transformed (in-place). New OHE cols: 16
Final shapes: (32136, 34) (10700, 34) (10666, 34)
```

In [58]:
```
# Provide some explanations on why you believe it is important to perform this data transform
data_transformation_3_explanations = """
Invalid values cannot be dropped without losing too much data. Replacing them with “Missing” I
One-hot encoding converts categorical features into a numerical format without imposing order
```

```
One-hot encoding then converts categorical features into a machine-readable form, ensuring co
"""
```

In [59]:
```
# DO NOT MODIFY THE CODE IN THIS CELL
print_tile(size="h3", key='data_transformation_3_explanations', value=data_transformation_3_e
```

data_transformation_3_explanations

Invalid values cannot be dropped without losing too much data. Replacing them with “Missing” keeps all records. One-hot encoding converts categorical features into a numerical format without imposing order. One-hot encoding then converts categorical features into a machine-readable form, ensuring consistent inputs and improving model performance.

E. Save Datasets

> Do not change this code

In [60]:
```
# DO NOT MODIFY THE CODE IN THIS CELL

try:
  X_train.to_csv(at.folder_path / 'X_train.csv', index=False)
  y_train.to_csv(at.folder_path / 'y_train.csv', index=False)

  X_val.to_csv(at.folder_path / 'X_val.csv', index=False)
  y_val.to_csv(at.folder_path / 'y_val.csv', index=False)

  X_test.to_csv(at.folder_path / 'X_test.csv', index=False)
  y_test.to_csv(at.folder_path / 'y_test.csv', index=False)
except Exception as e:
  print(e)
```