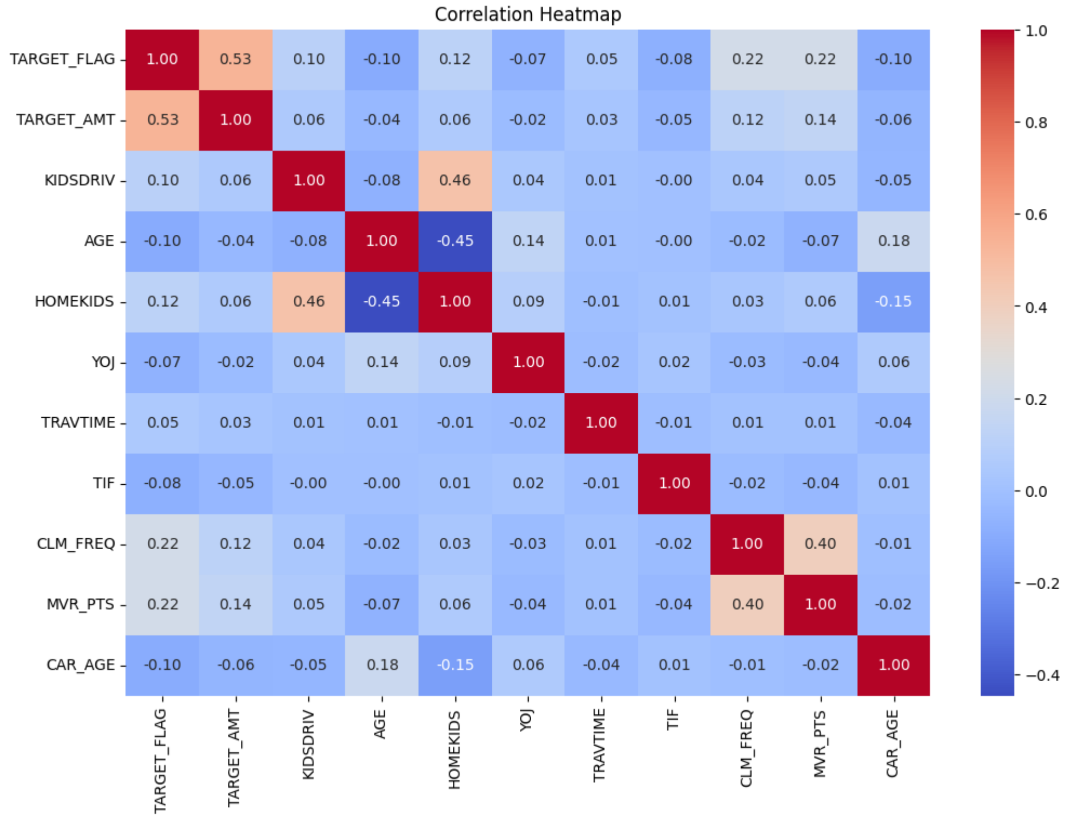
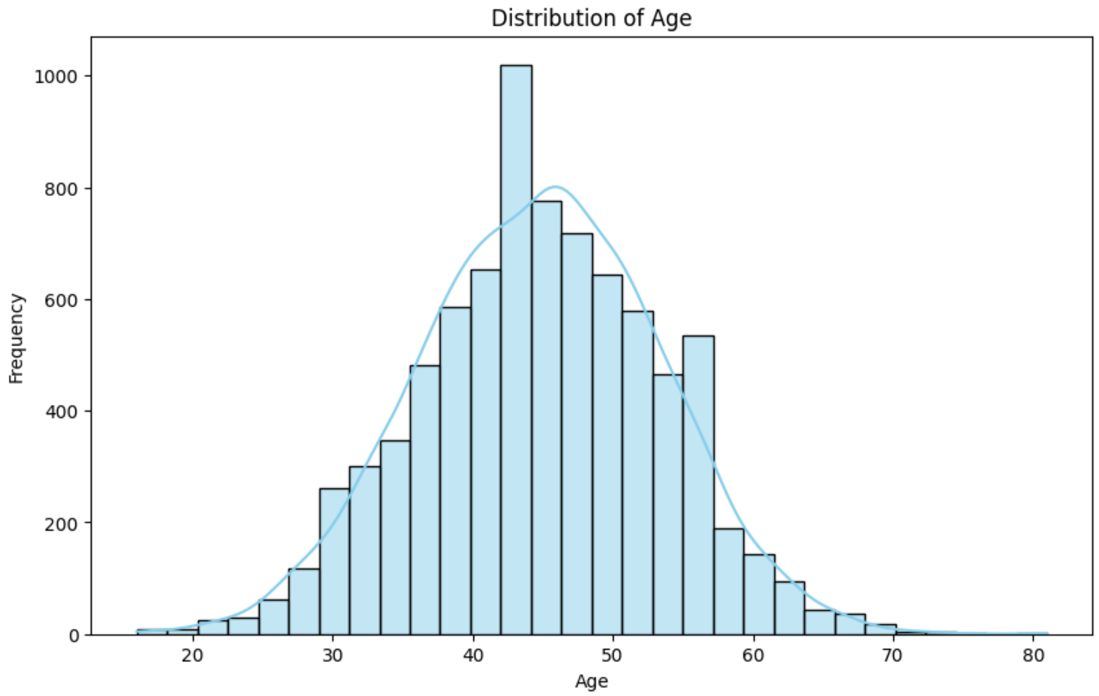
HW5

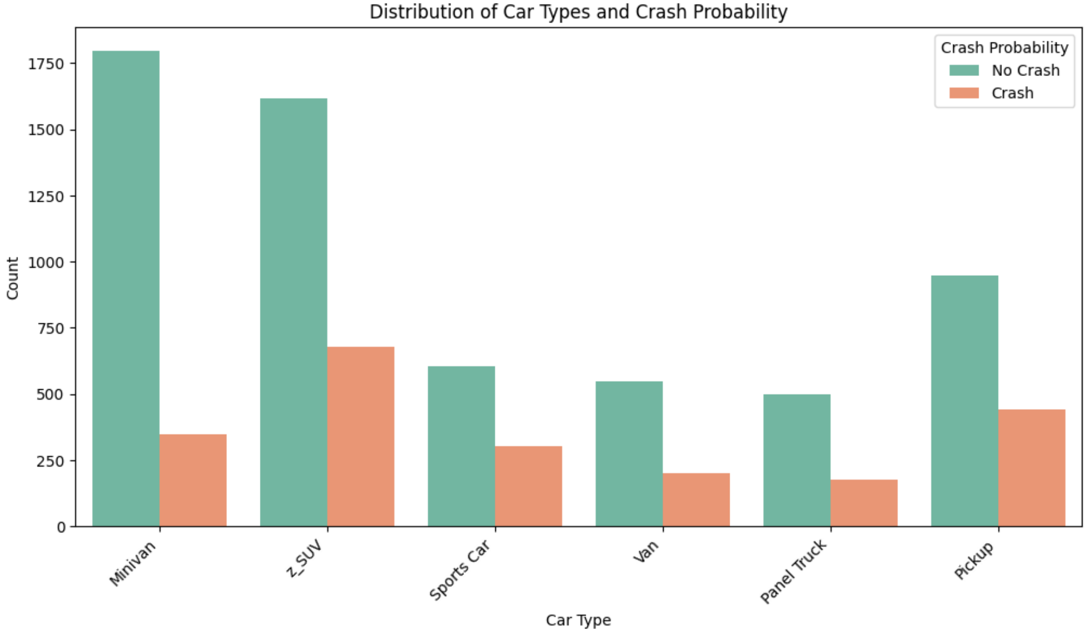
Keeno Glanville

Data Exploration

The data exploration was quite telling, it was able to alleviate personal expectations about age distribution. I would assume more youth would be involved in the crashes. The data was also not strongly correlated which would be intuitive seeing as no one can really predict a crash







Data Preparation

```
TARGET_FLAG      0
TARGET_AMT       0
KIDSDRIV         0
AGE              6
HOMEKIDS         0
YOJ            454
INCOME         445
PARENT1          0
HOME_VAL       464
MSTATUS          0
SEX              0
EDUCATION        0
JOB            526
TRAVTIME         0
CAR_USE          0
BLUEBOOK         0
TIF              0
CAR_TYPE         0
RED_CAR          0
OLDCLAIM         0
CLM_FREQ         0
REVOKED          0
MVR_PTS          0
CAR_AGE        510
URBANICITY       0
dtype: int64
```

```
TARGET_FLAG    0
TARGET_AMT     0
KIDSDRIV       0
AGE            0
HOMEKIDS       0
YOJ            0
INCOME         0
PARENT1        0
HOME_VAL       0
MSTATUS        0
SEX            0
EDUCATION      0
JOB            0
TRAVTIME       0
CAR_USE        0
BLUEBOOK       0
TIF            0
CAR_TYPE       0
RED_CAR        0
OLDCLAIM       0
CLM_FREQ       0
REVOKED        0
MVR_PTS        0
CAR_AGE        0
URBANICITY     0
dtype: int64
```

The data preparation consisted of fixing typographical inconsistencies as well as imputing missing values. For the most part the dataset was easy to work with.

```python
# Convert object columns to numeric
train['INCOME'] = pd.to_numeric(train['INCOME'].str.replace('[\$,]', ''), errors='coerce')
train['HOME_VAL'] = pd.to_numeric(train['HOME_VAL'].str.replace('[\$,]', ''), errors='coerce'
train['BLUEBOOK'] = pd.to_numeric(train['BLUEBOOK'].str.replace('[\$,]', ''), errors='coerce'
train['OLDCLAIM'] = pd.to_numeric(train['OLDCLAIM'].str.replace('[\$,]', ''), errors='coerce'
train['MSTATUS'] = train['MSTATUS'].replace('z_No', 'No')
train['SEX'] = train['SEX'].replace('z_F', 'F')

# Convert object columns to numeric
test['INCOME'] = pd.to_numeric(test['INCOME'].str.replace('[\$,]', ''), errors='coerce')
test['HOME_VAL'] = pd.to_numeric(test['HOME_VAL'].str.replace('[\$,]', ''), errors='coerce')
test['BLUEBOOK'] = pd.to_numeric(test['BLUEBOOK'].str.replace('[\$,]', ''), errors='coerce')
test['OLDCLAIM'] = pd.to_numeric(test['OLDCLAIM'].str.replace('[\$,]', ''), errors='coerce')
test['MSTATUS'] = test['MSTATUS'].replace('z_No', 'No')
test['SEX'] = test['SEX'].replace('z_F', 'F')


# Print updated column types
column_types = train.dtypes
print(column_types)
```

Build Models

```
# Define features and target
X_flag = train.drop(['TARGET_FLAG', 'TARGET_AMT'], axis=1)
y_flag = train['TARGET_FLAG']

# Convert categorical variables to dummies
X_flag = pd.get_dummies(X_flag, drop_first=True)

# Split the data into training and testing sets
X_train_flag, X_test_flag, y_train_flag, y_test_flag = train_test_split(X_flag, y_flag, test_size=0.2, random_state=42)

# Binary Logistic Regression
log_reg_flag = LogisticRegression(random_state=42)
log_reg_flag.fit(X_train_flag, y_train_flag)

# Evaluate the model
y_pred_flag = log_reg_flag.predict(X_test_flag)
print(confusion_matrix(y_test_flag, y_pred_flag))
print(classification_report(y_test_flag, y_pred_flag))
```

```
[[1158   31]
 [ 418   26]]
              precision    recall  f1-score   support

           0       0.73      0.97      0.84      1189
           1       0.46      0.06      0.10       444

    accuracy                           0.73      1633
   macro avg       0.60      0.52      0.47      1633
weighted avg       0.66      0.73      0.64      1633
```

```
# Define features and target
X_amt = train.drop(['TARGET_FLAG', 'TARGET_AMT'], axis=1)
y_amt = train['TARGET_AMT']

# Convert categorical variables to dummies
X_amt = pd.get_dummies(X_amt, drop_first=True)

# Split the data into training and testing sets
X_train_amt, X_test_amt, y_train_amt, y_test_amt = train_test_split(X_amt, y_amt, test_size=0.2, random_state=42)

# Linear Regression for predicting continuous variable
lin_reg_amt = LinearRegression()
lin_reg_amt.fit(X_train_amt, y_train_amt)

# Evaluate the model
y_pred_amt = lin_reg_amt.predict(X_test_amt)
print('Mean Squared Error:', mean_squared_error(y_test_amt, y_pred_amt))
print('R-squared:', r2_score(y_test_amt, y_pred_amt))
```

```
Mean Squared Error: 28992700.242169943
R-squared: 0.061593652624510664
```

Select Models

Model was tuned to select the most optimal features to present a more accurate model, however the model had a very difficult time fitting the data

```
# Linear Regression with Feature Selection
lin_reg_amt = LinearRegression()

# Use SelectFromModel for feature selection
feature_selector_amt = SelectFromModel(lin_reg_amt)
X_train_amt_selected = feature_selector_amt.fit_transform(X_train_amt, y_train_amt)
X_test_amt_selected = feature_selector_amt.transform(X_test_amt)

# Train the model with selected features
lin_reg_amt.fit(X_train_amt_selected, y_train_amt)

# Evaluate the model
y_pred_amt = lin_reg_amt.predict(X_test_amt_selected)
print('Mean Squared Error:', mean_squared_error(y_test_amt, y_pred_amt))
print('R-squared:', r2_score(y_test_amt, y_pred_amt))
```

```
Mean Squared Error: 29467483.956893705
R-squared: 0.04622633437523449
```