Autozen Explorarotry Data Analysis (EDA)

Welcome to the Exploratory Data Analysis (EDA) notebook for the dataset. This notebook aims to provide a comprehensive analysis of the preprocessed dataset we pulled from the databases, uncovering patterns, relationships, and insights that can help us better understand the data. EDA is an essential step in the data analysis process, as it allows us to gain familiarity with the dataset and derive meaningful conclusions.

In this notebook, we will perform a variety of exploratory data analysis techniques to gain insights into the dataset. EDA involves examining the structure and content of the data, identifying missing values, outliers, and anomalies, as well as investigating the relationships between variables. By visualizing the data and conducting statistical analysis, we can uncover patterns, trends, and correlations that can guide us in further analysis or modeling tasks.

The objectives of this EDA notebook include:

1. Data Familiarization: We will start by getting an overview of the dataset, including its size, variables, and their types. Understanding the data's structure is crucial for subsequent analysis.
2. Descriptive Statistics: We will compute various descriptive statistics such as mean, median, standard deviation, and quartiles to summarize the central tendencies and spread of the variables. This will provide us with a better understanding of the dataset's distribution.
3. Data Cleaning: Before diving into the analysis, we will address data cleaning tasks such as handling missing values, removing duplicates, and addressing outliers. This step ensures the reliability and accuracy of our analysis.
4. Univariate Analysis: We will explore each variable individually, visualizing their distributions using histograms, box plots, or bar graphs. This analysis will help us identify any patterns, anomalies, or outliers within individual variables.
5. Bivariate and Multivariate Analysis: We will investigate the relationships between variables using scatter plots, correlation matrices, or pair plots. This analysis will enable us to uncover associations, dependencies, or trends between different variables in the dataset.
6. Key Findings and Insights: Based on our analysis, we will summarize the key findings, insights, and patterns discovered during the EDA process. These insights can guide further analysis or decision-making processes.

For this dataset, we have a wide range of variables capturing various aspects of the vehicles, including general information, exterior condition, tire condition, under-vehicle components, interior features, lighting, and test drive observations. Each variable will be explored and analyzed to extract valuable insights.

Load Data and Libraries

```
# all reusable code is located in scripts and no in Jupyter Notebooks
import sys
import os
import datetime

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.graph_objects as go
import plotly.subplots as sp
import plotly.io as pio
import plotly.express as px
pio.renderers.default = 'notebook'

sys.path.append('./scripts')
notebook_dir = os.getcwd()
data_dir = os.path.join(notebook_dir, ".", "data")

from data_collection import DataCollector
from autozen_features import AutozenFeatures
from model_ml_training import ModelTrainerML
from model_nn_training import ModelTrainerNN
from utils import get_column_info

import warnings
warnings.filterwarnings("ignore")
```

```
# Autozen features are defined in AutozenFeatures in the scripts modules
MERGED_PROCESSED_WON = os.path.join(data_dir,"processed_az_auctioned_won.csv")
MERGED_PROCESSED_AUCTIONED = os.path.join(data_dir,"processed_az_auctioned.csv")
df_won = pd.read_csv(MERGED_PROCESSED_WON)
df_auctioned = pd.read_csv(MERGED_PROCESSED_AUCTIONED)
df_won = df_won.drop(AutozenFeatures.to_drop_feat_list,axis=1)
print(f"new shape after dropping columns not required {df_won.shape}")
```

```
new shape after dropping columns not required (1323, 213)
```

Dealing with Missing Data

We start with examining the dataset and checking for missing values within different columns. Understanding the extent and patterns of missing values is crucial for ensuring data integrity and making informed decisions about data handling and preprocessing. To begin with, we perform a comprehensive examination of the dataset to identify the presence and distribution of missing values across different variables. Various visualizations are employed to provide a clear understanding of the missing data patterns. These visualizations include:

1. Visual Inspection:
2. Missing Data Matrix: We construct a matrix that displays the presence or absence of data for each variable in the dataset. Missing values are represented by blank cells or a distinct color, while non-missing values are denoted by filled cells or another color. This matrix provides an overview of the missingness across all variables, enabling us to identify variables with high or low missing data.
3. Missing Data Bar plot: This plot illustrates the percentage of missing values for each variable in the dataset. It provides a concise summary of the missing data distribution, highlighting variables that may require special attention or further investigation.

Therefore, What can we say about the missing data in Autozen training data set ? The Autozen data has a very high percentage of missing data values that we would need to impute or process in different ways before we input this into the machine learning models.

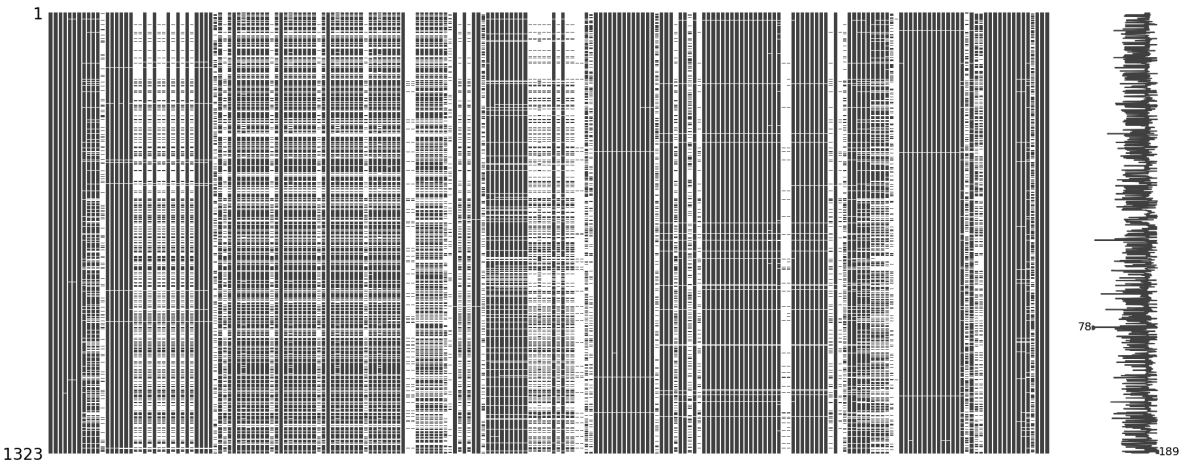
```
columns_list = get_column_info(df_won)
columns_list.sort_values("Null Values", ascending=False).head(10)
```

```
(1323, 213)
(213, 4)
```

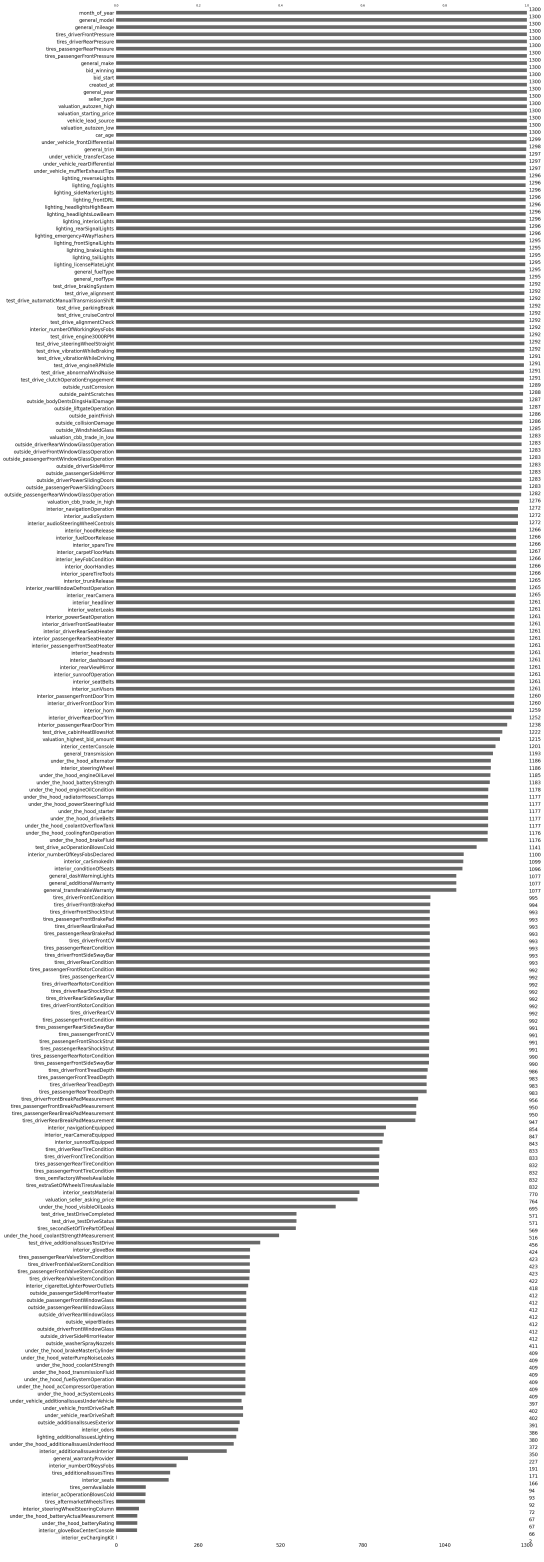
	Column	Unique Values	Null Values	Type
180	interior_evChargingKit	1	1321	object
156	interior_gloveBoxCenterConsole	2	1257	object
112	under_the_hood_batteryRating	26	1256	float64
113	under_the_hood_batteryActualMeasurement	49	1256	float64
157	interior_steeringWheelSteeringColumn	2	1251	object
76	tires_aftermarketWheelsTires	2	1231	object
168	interior_acOperationBlowsCold	2	1230	object
77	tires_oemAvailable	2	1229	object
138	interior_seats	2	1157	object
85	tires_additionalIssuesTires	3	1149	object

```
# Visualize missing values as a matrix
plt.figure(figsize=(16, 9))  # Adjust the size as needed
msno.matrix(df_won)
plt.show()
```

```
<Figure size 1600x900 with 0 Axes>
```




```
missing_values = df_won.isnull().sum()
missing_values_sorted = missing_values.sort_values(ascending=False)
sorted_columns = missing_values_sorted.index
df_sorted = df_won[sorted_columns]
msno.bar(df_sorted.sample(1300))
plt.show()
```


month_of_year 1300
general_model 1300
general_mileage 1300
tires_driverFrontPressure 1300
tires_driverRearPressure 1300
tires_passengerRearPressure 1300
tires_passengerFrontPressure 1300
general_make 1300
bid_winning 1300
bid_start 1300
created_at 1300
general_year 1300
seller_type 1300
valuation_autozen_high 1300
valuation_starting_price 1300
vehicle_lead_source 1300
valuation_autozen_low 1300
car_age 1299
under_vehicle_frontDifferential 1298
general_trim 1298
under_vehicle_transferCase 1297
under_vehicle_rearDifferential 1297
under_vehicle_muffler&ExhaustTips 1296
lighting_reverseLights 1296
lighting_fogLights 1296
lighting_sideMarkerLights 1296
lighting_frontDRL 1296
lighting_headlightsHighBeam 1296
lighting_headlightsLowBeam 1296
lighting_interiorLights 1296
lighting_rearSignalLights 1296
lighting_emergency4WayFlashers 1296
lighting_frontSignalLights 1295
lighting_brakeLights 1295
lighting_tailLights 1295
lighting_licensePlateLight 1295
general_fuelType 1295
general_roofType 1295
test_drive_brakingSystem 1292
test_drive_alignment 1292
test_drive_automaticManualTransmissionShift 1292
test_drive_parkingBreak 1292
test_drive_cruiseControl 1292
test_drive_alignmentCheck 1292
interior_numberOfWorkingKeysValet 1292
test_drive_engine3000RPM 1292
test_drive_steeringWheel50orright 1292
test_drive_vibrationsWhileBraking 1291
test_drive_vibrationsWhileDriving 1291
test_drive_engineRPMidle 1291
test_drive_abnormalWindNoise 1291
test_drive_clutchOperationEngagement 1289
outside_rustCorrosion 1288
outside_paintScratches 1287
outside_bodyDents2inchesDamage 1287
outside_liftgateOperation 1286
outside_paintFinish 1286
outside_collisionDamage 1285
outside_WindshieldGlass 1283
valuation_ctib_trade_in_low 1283
outside_driverRearWindowGlassOperation 1283
outside_driverFrontWindowGlassOperation 1283
outside_passengerFrontWindowGlassOperation 1283
outside_driverSideMirror 1283
outside_passengerSideMirror 1283
outside_driverPowerSlidingDoors 1283
outside_passengerPowerSlidingDoors 1283
outside_passengerRearWindowGlassOperation 1282
valuation_ctib_trade_in_high 1276
interior_navigationOperation 1272
interior_audioSystem 1272
interior_audioSteeringWheelControls 1272
interior_hoodRelease 1266
interior_fuelDoorRelease 1266
interior_spareTire 1266
interior_carpetFloorMats 1267
interior_keyFobsCondition 1266
interior_doorHandles 1266
interior_spareTireTools 1266
interior_trunkRelease 1265
interior_rearWindowDefrostOperation 1265
interior_rearCamera 1265
interior_headliner 1261
interior_waterLeaks 1261
interior_powerSeatOperation 1261
interior_driverFrontSeatHeater 1261
interior_driverRearSeatHeater 1261
interior_passengerRearSeatHeater 1261
interior_passengerFrontSeatHeater 1261
interior_headrests 1261
interior_dashboard 1261
interior_rearViewMirror 1261
interior_sunroofOperation 1261
interior_seatBelts 1261
interior_sunVisors 1261
interior_passengerFrontDoorTrim 1260
interior_driverFrontDoorTrim 1260
interior_horn 1259
interior_driverRearDoorTrim 1252
interior_passengerRearDoorTrim 1238
test_drive_cabinHeatEffanHot 1222
valuation_highest_bid_amount 1215
interior_centerConsole 1201
general_transmission 1193
under_the_hood_alternator 1186
interior_steeringWheel 1186
under_the_hood_engineOilLevel 1185
under_the_hood_batteryStrength 1183
under_the_hood_engineOilCondition 1178
under_the_hood_radiatorHosesClamps 1177
under_the_hood_powerSteeringFluid 1177
under_the_hood_starter 1177
under_the_hood_driveBelts 1177
under_the_hood_coolantOverflowTank 1177
under_the_hood_coolingFanOperation 1176
under_the_hood_brakeFluid 1176
test_drive_acOperationBlowsCold 1141
interior_numberOfKeysFobsDeclared 1100
interior_carSmokeIndn 1099
interior_conditionOfSeats 1066
general_dashWarningLights 1077
general_additionalWarranty 1077
general_transferableWarranty 1077
tires_driverFrontCondition 995
tires_driverFrontBrakePad 994
tires_driverFrontShockStrut 993
tires_passengerFrontBrakePad 993
tires_driverRearBrakePad 993
tires_passengerRearBrakePad 993
tires_driverFrontTV 993
tires_passengerRearCondition 993
tires_driverFrontSideSwayBar 993
tires_driverRearCondition 993
tires_passengerFrontRotorCondition 992
tires_passengerRearCV 992
tires_driverRearRotorCondition 992
tires_driverRearShockStrut 992
tires_driverRearSideSwayBar 992
tires_driverFrontRotorCondition 992
tires_driverRearCV 992
tires_passengerFrontCondition 992
tires_passengerRearSideSwayBar 991
tires_passengerFrontCV 991
tires_passengerFrontShockStrut 991
tires_passengerRearShockStrut 991
tires_passengerRearRotorCondition 990
tires_passengerFrontSideSwayBar 990
tires_driverFrontTreadDepth 986
tires_passengerFrontTreadDepth 983
tires_driverRearTreadDepth 983
tires_passengerRearTreadDepth 983
tires_driverFrontBreakPadsMeasurement 960
tires_passengerFrontBreakPadsMeasurement 956
tires_passengerRearBreakPadsMeasurement 950
tires_driverRearBreakPadsMeasurement 950
interior_navigationEquipped 947
interior_rearCameraEquipped 854
interior_sunroofEquipped 847
tires_driverFrontTireCondition 843
tires_driverRearTireCondition 833
tires_passengerRearTireCondition 833
tires_passengerFrontTireCondition 832
tires_oemFactoryWheelsAvailable 832
tires_extraSetOfWheelsTiresAvailable 832
interior_seatsMaterial 822
valuation_seller_asking_price 770
under_the_hood_visibleOilLeaks 764
test_drive_testDriveCompleted 695
test_drive_testDriveStatus 571
tires_secondSetOfTiresNotOfDeal 571
under_the_hood_coolantStrengthMeasurement 569
test_drive_additionalIssuesTestDrive 316
interior_gloveBox 456
tires_passengerRearValveStemCondition 424
tires_driverFrontValveStemCondition 423
tires_passengerFrontValveStemCondition 423
tires_driverRearValveStemCondition 423
interior_cigaretteLighterPowerOutlets 422
outside_passengerSideMirrorHeater 418
outside_passengerFrontWindowGlass 412
outside_passengerRearWindowGlass 412
outside_driverRearWindowGlass 412
outside_wiperBlades 412
outside_driverFrontWindowGlass 412
outside_driverSideMirrorHeater 412
outside_washerSprayNozzels 412
under_the_hood_brakeMasterCylinder 411
under_the_hood_waterPumpNoiseLeaks 409
under_the_hood_coolantStrength 409
under_the_hood_transmissionFluid 409
under_the_hood_fuelSystemsOperation 409
under_the_hood_acCompressorOperation 409
under_the_hood_acSystemLeaks 409
under_vehicle_additionalIssuesUnderVehicle 397
under_vehicle_frontDriveShaft 402
under_vehicle_rearDriveShaft 402
outside_additionalIssuesExterior 391
interior_odors 386
lighting_additionalIssuesLighting 380
under_the_hood_additionalIssuesUnderHood 372
interior_additionalIssuesInterior 350
general_warrantyProvider 227
interior_numberOfNoisyKnobs 191
tires_additionalIssuesTires 171
interior_seats 166
tires_aervaAvailable 96
interior_acOperationBlowsCold 95
tires_aftermarketWheelsTires 92
interior_steeringWheelSteeringColumn 72
under_the_hood_batteryActualMeasurement 67
under_the_hood_batteryRating 67
interior_gloveBoxCenterConsole 66
interior_evChargingKit 1
0
260
520
780
1040
1300

Based on the missing value data matrix and the bar chart we see a great majority of columns with missing data or very few entries as well. To deal with these there are two approaches we might take, one is dropping the columns entirely and the other is using an imputation method like k-nn imputation for instance.

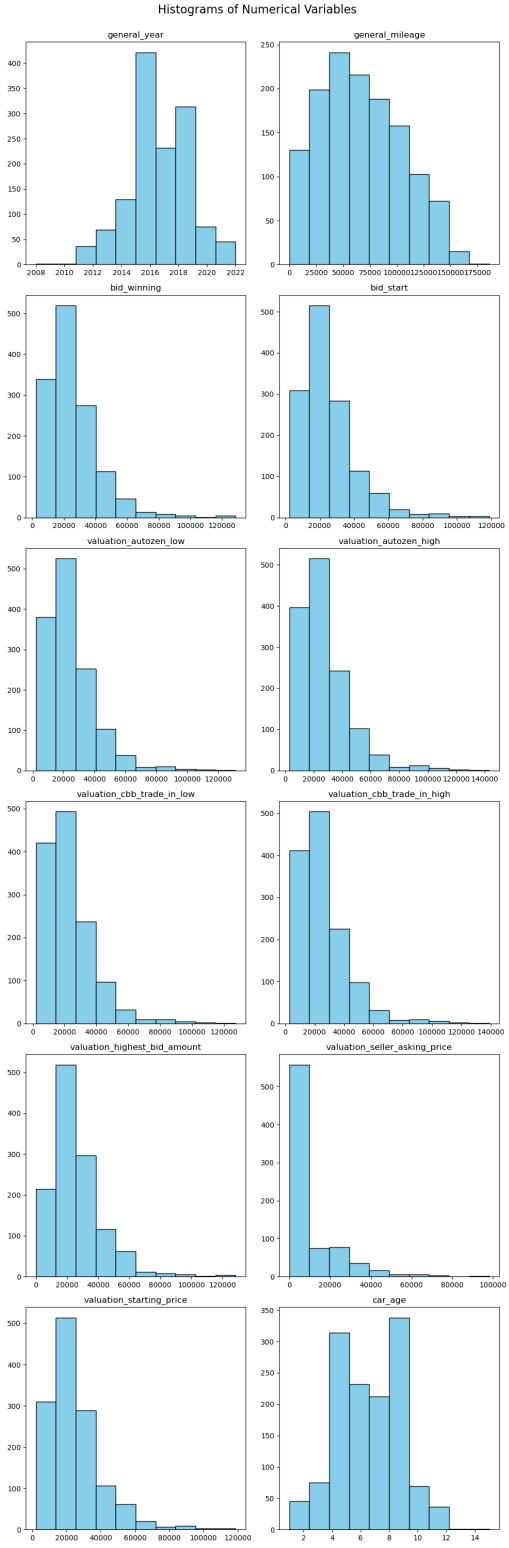
Visualizing Numerical Variables

1. **Histograms**: provide a visual representation of the distribution of a single variable. By dividing the range of values into intervals or bins and counting the number of observations falling into each bin, histograms help us understand the frequency and density of values within the variable. In our analysis, we created histograms for each numerical variable, allowing us to examine their shape, central tendency, and spread. This visualization enabled us to identify any outliers, assess the presence of skewness, and get an overall understanding of the data distribution.
2. **Pairplots**: on the other hand, offer a comprehensive view of the relationships between multiple numerical variables. By plotting pairwise scatter plots of variables against each other, pairplots allow us to observe the potential correlations, trends, and patterns in the data. This visualization is particularly useful when exploring the interactions between different variables and looking for potential associations. In our analysis, we generated a pairplot for the numerical variables in the dataset, which provided us with a matrix of scatter plots. This allowed us to examine the pairwise relationships and identify any apparent connections or dependencies between the variables.
3. **Box plots**: also known as box-and-whisker plots, provide a visual summary of the distribution of a numerical variable. They display the median, quartiles, and potential outliers, allowing for a quick assessment of the spread, skewness, and presence of extreme values. By drawing a box that represents the interquartile range (IQR) and "whiskers" that extend to the minimum and maximum values within a certain range, box plots provide a concise representation of the data distribution and help identify any deviations from the norm. In our analysis, we constructed box plots for each numerical variable, enabling us to compare their central tendencies, spread, and variability. This visualization allowed us to quickly identify outliers, assess the presence of skewed distributions, and understand the overall range of values for each variable.
4. **Correlation Matrix:** Another important visualization for numerical variables is the correlation matrix. The correlation matrix provides a comprehensive view of the relationships between pairs of variables by displaying the correlation coefficients between them. Correlation coefficients measure the strength and direction of the linear relationship between two variables, ranging from -1 (strong negative correlation) to 1 (strong positive correlation), with 0 indicating no correlation.

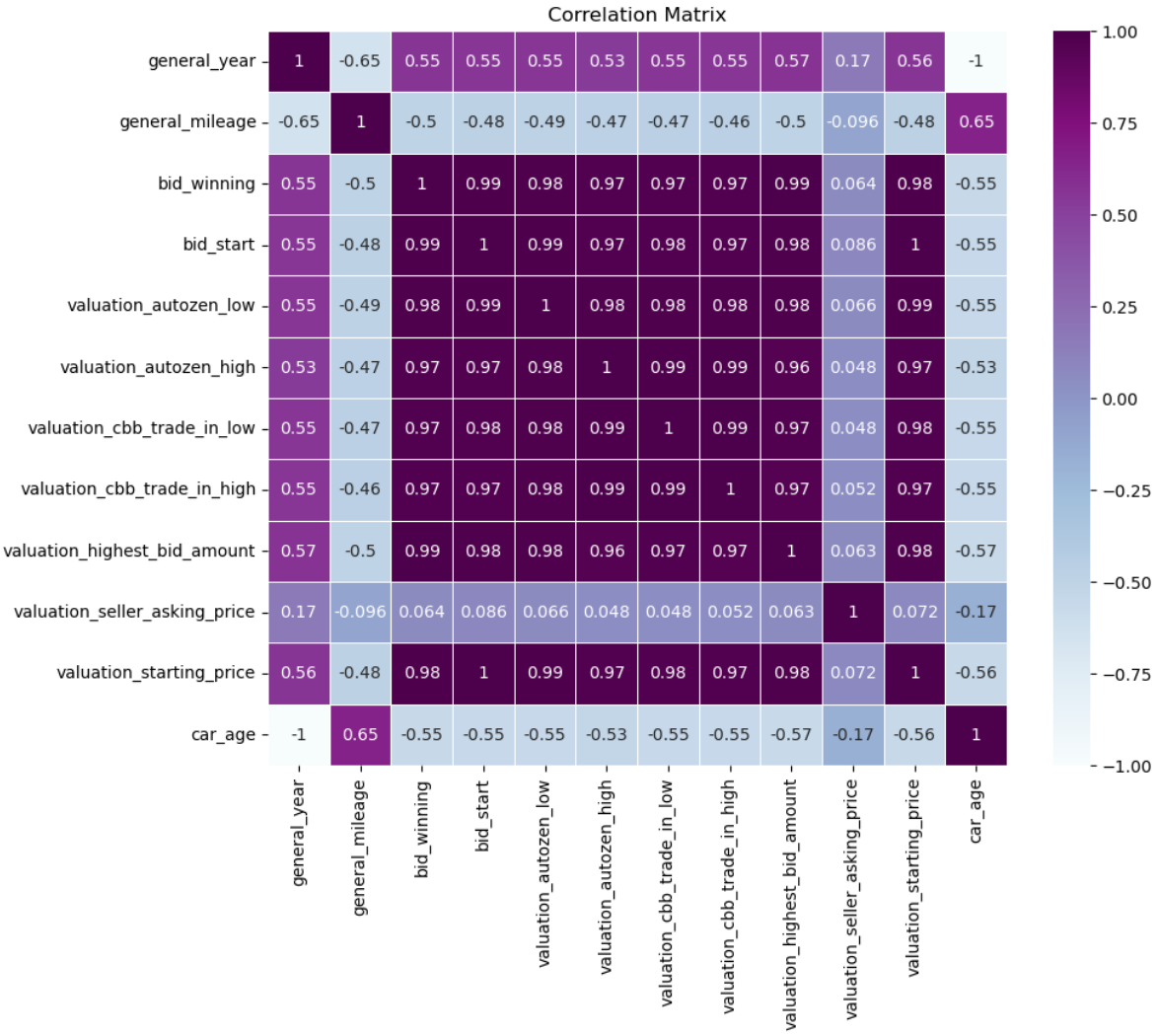
Together, histograms, boxplots, correlation matrices, and pairplots provide valuable visualizations to understand the distribution, relationship, and behavior of numerical variables in the

dataset. These insights can guide further analysis, feature selection, or modeling decisions, helping us gain a deeper understanding of the dataset and its underlying characteristics.

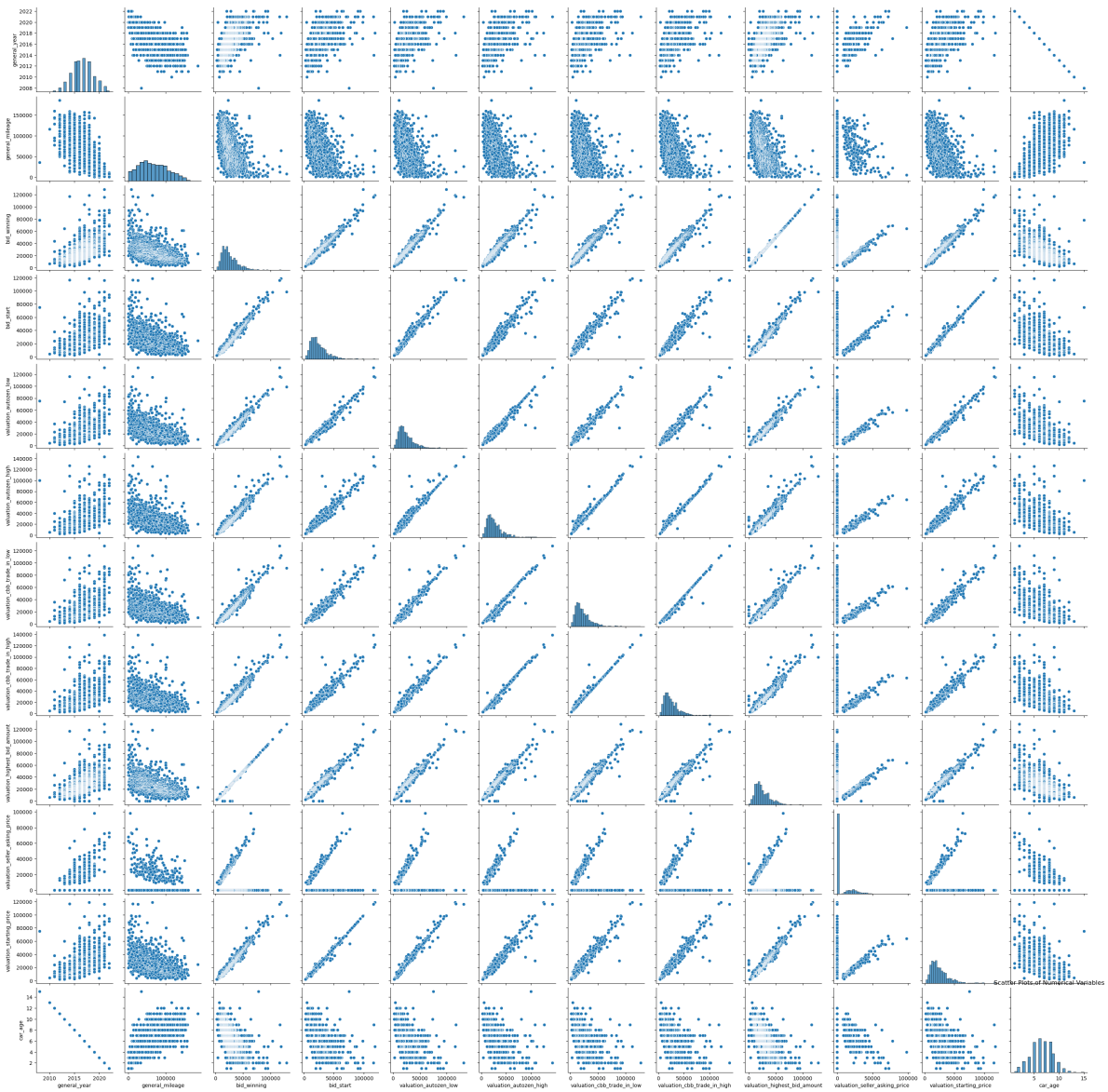
```
# Create dataframe with numerical variables
num_df = df_won[['general_year', 'general_mileage', 'bid_winning', 'bid_start',
                 'valuation_autozen_low', 'valuation_autozen_high', 'valuation_cbb_trade_i
       'valuation_cbb_trade_in_high', 'valuation_highest_bid_amount', 'valuation_seller_as
       'valuation_starting_price', 'car_age']]

num_rows = 6
fig, axes = plt.subplots(num_rows, 2, figsize=(10, num_rows*5))
fig.suptitle('Histograms of Numerical Variables', fontsize=16, y=1.0)
axes = axes.flatten()

for i, col in enumerate(num_df.columns):
    ax = axes[i]
    ax.hist(num_df[col].dropna(), color='skyblue', edgecolor='black')
    ax.set_title(col)
fig.tight_layout()
plt.show()
```


Histograms of Numerical Variables
general_year
general_mileage
bid_winning
bid_start
valuation_autozen_low
valuation_autozen_high
valuation_cbb_trade_in_low
valuation_cbb_trade_in_high
valuation_highest_bid_amount
valuation_seller_asking_price
valuation_starting_price
car_age

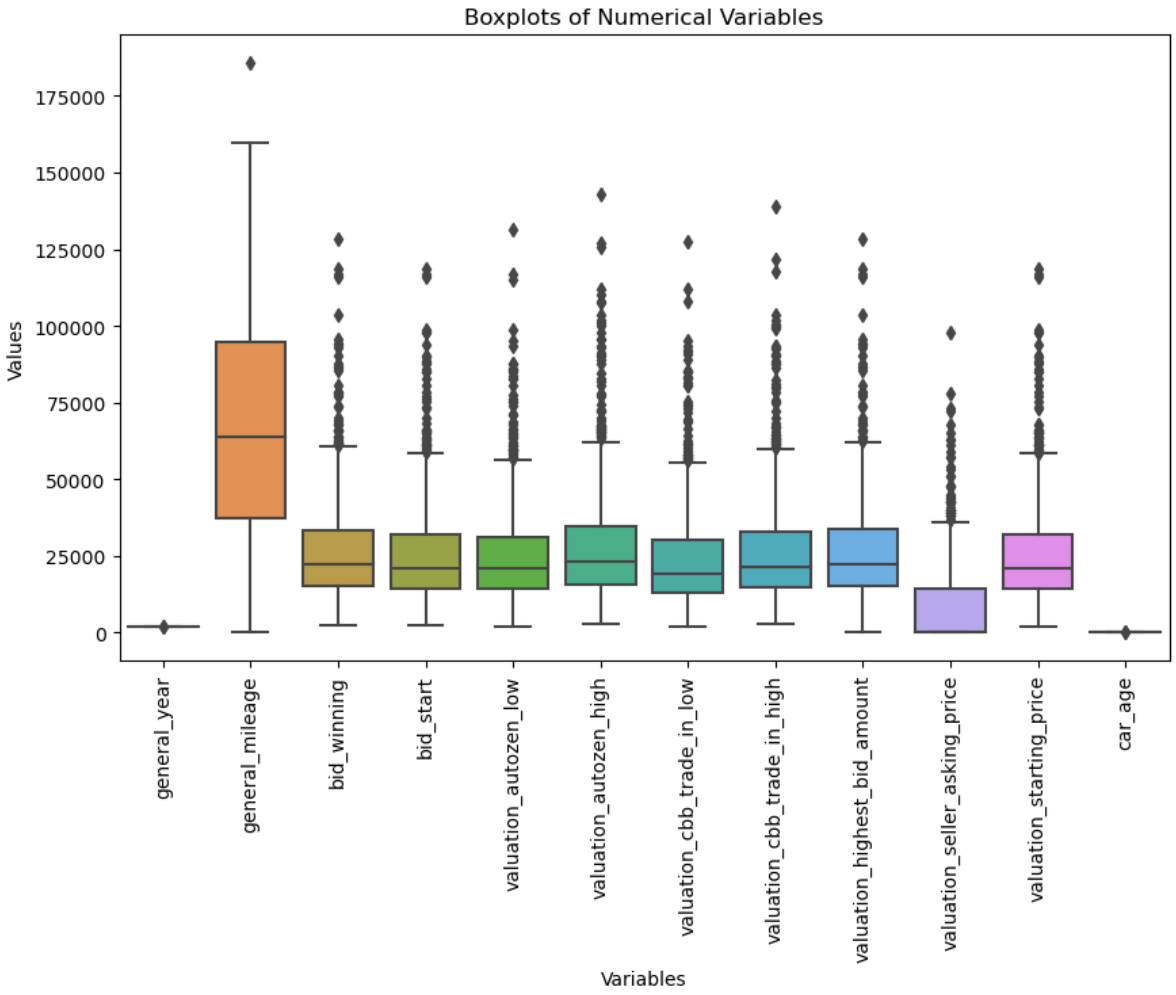
```
# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(num_df.corr(), annot=True, cmap='BuPu', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```




```
sns.pairplot(data=num_df, kind='scatter')
plt.title('Scatter Plots of Numerical Variables')
plt.show()
```




```
fig, axes = plt.subplots(figsize=(10, 6))
sns.boxplot(data=num_df, ax=axes)
axes.set_title('Boxplots of Numerical Variables')
axes.set_xlabel('Variables')
axes.set_ylabel('Values')
axes.set_xticklabels(axes.get_xticklabels(), rotation=90)
plt.show()
```




```
# Interactive Box Plots
fig = go.Figure()
for col in num_df.columns:
    fig.add_trace(go.Box(y=num_df[col], name=col))

fig.update_layout(
    title='Boxplots of Numerical Variables',
    xaxis_title='Variables',
    yaxis_title='Values',
    xaxis=dict(tickangle=90)
)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

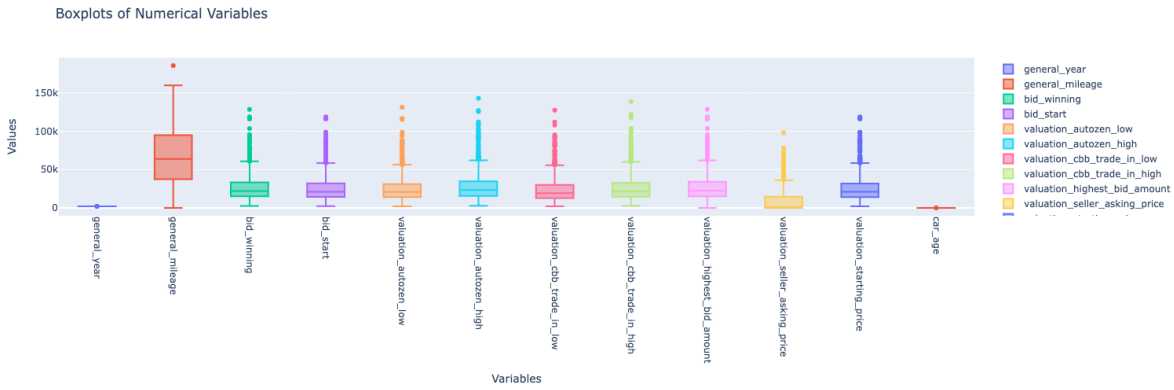


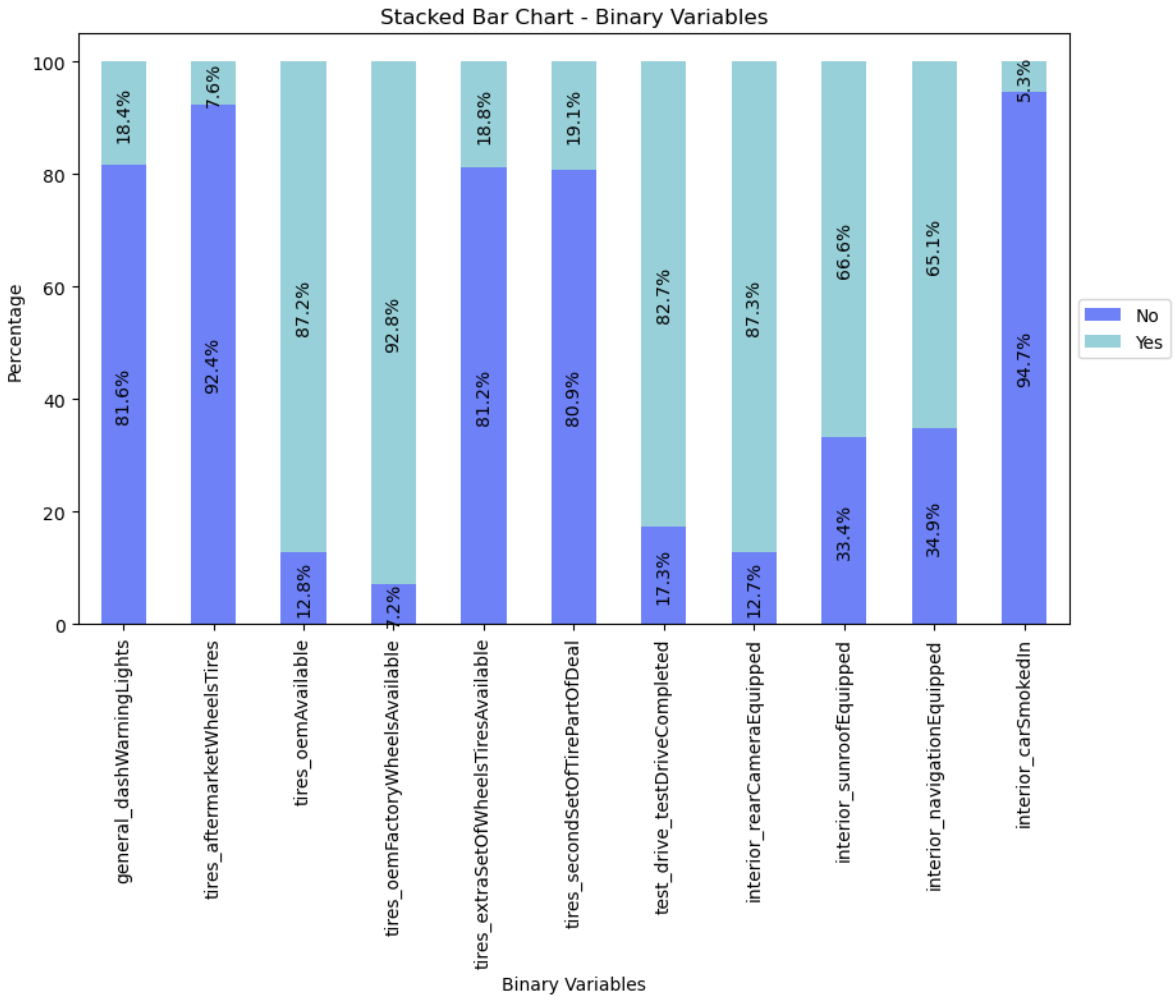
Figure 1: boxplot

Visualizing Binary Variables

Visualizing the distribution percentages of binary variables is valuable in exploratory data analysis (EDA) as it provides insights into the prevalence and balance of different categories within each variable. By displaying the percentages on a stacked bar chart, we can quickly observe the relative frequency of "Yes" and "No" responses. This visualization helps us understand the distribution patterns and identify any imbalances or biases in the dataset. It allows us to assess the proportion of positive and negative instances, which can be crucial in scenarios where class imbalance may affect model performance or decision-making. Additionally, visualizing the binary variables' distribution percentages aids in detecting any potential data quality issues, such as missing or erroneous values, and guides further data preprocessing and feature engineering steps.

```
df_binary = df_won[AutozenFeatures.binary_feat_list]
binary_counts = df_binary.apply(lambda x: x.value_counts(normalize=True) * 100)
binary_counts = binary_counts.transpose()
colors = ['#6e81f7', '#97D0D9']
ax = binary_counts.plot(kind='bar', stacked=True, figsize=(10, 6), color=colors)

# Display percentage values on the bars
for p in ax.patches:
    width = p.get_width()
```

```
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.1f}%', (x + width/2, y + height/2), ha='center', va='center', r

plt.title('Stacked Bar Chart - Binary Variables')
plt.xlabel('Binary Variables')
plt.ylabel('Percentage')
plt.legend(['No', 'Yes'], loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



Visualizing Categorical Variables

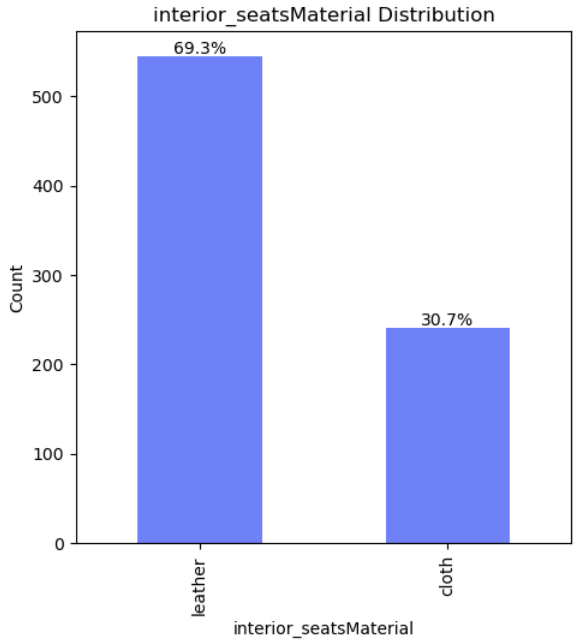
Visualizing categorical variables is essential during the exploratory data analysis (EDA) process and plays a crucial role in understanding the data and building an effective machine learning model. Categorical variable visualization helps us gain insights into the distribution, frequencies, and relationships between different categories, which can provide valuable information for feature engineering, model selection, and interpretation.

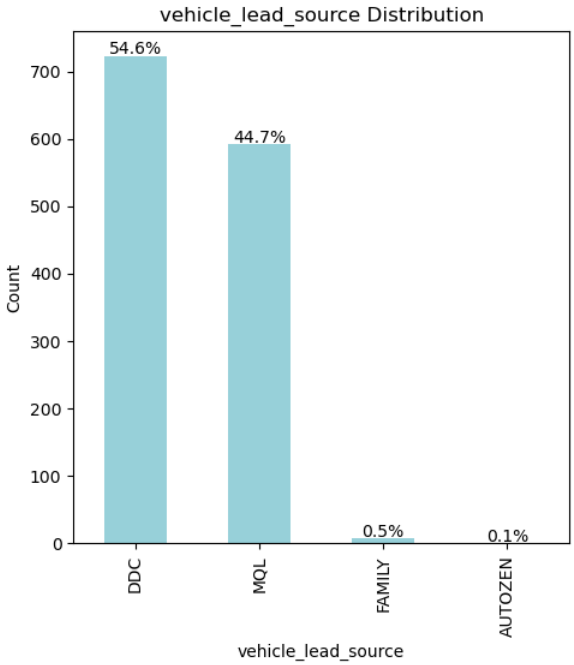
By visualizing categorical variables, we can identify the dominant categories and their proportions within each variable. This helps us understand the class imbalance, if any, and determine the representativeness of each category. Additionally, categorical variable visualizations allow us to observe patterns, trends, and associations between the categories, which can guide feature engineering decisions. For example, if certain categories have a higher winning bid on average, it might indicate their importance in predicting the winning bid. Visualizations also help us identify rare categories or outliers that may require special treatment during data preprocessing.

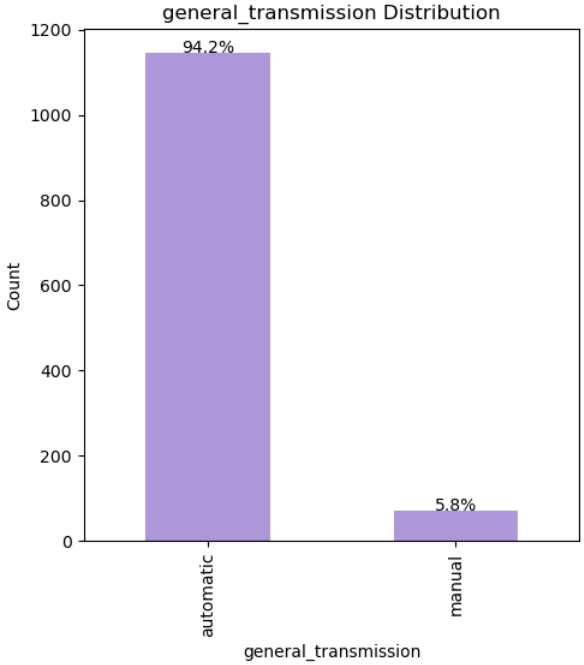
In the context of our final machine learning model with XGBoost to predict the winning bid for auctioned cars, analyzing and visualizing categorical variables can provide valuable insights. It helps us understand the impact of different categories on the target variable (winning bid) and uncover any relationships or dependencies. For instance, visualizing the "vehicle_lead_source" variable could reveal whether certain lead sources tend to result in higher or lower winning bids. This information can guide feature selection, model training, and tuning processes. By visualizing categorical variables, we can effectively explore the relationships between these variables and the target, aiding in the development of a robust and accurate predictive model.

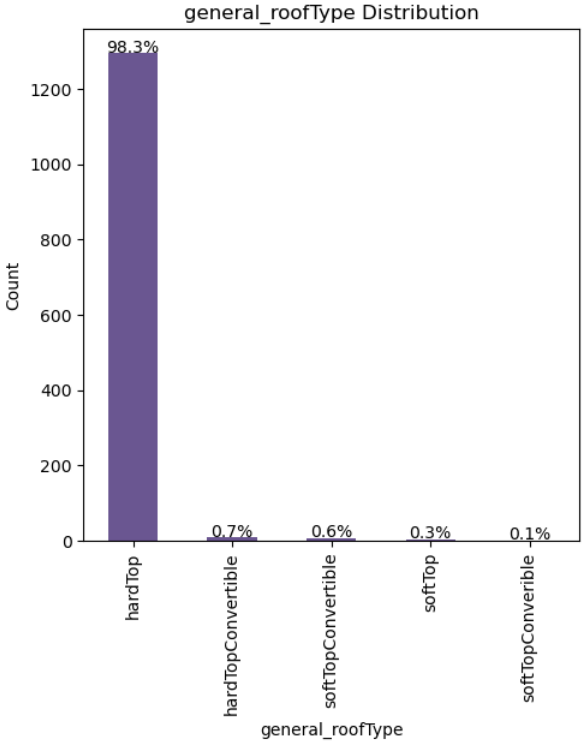
```
categorical_variables = ['interior_seatsMaterial', 'vehicle_lead_source', 'general_transmi
color = ['#6e81f7', '#97D0D9', '#AD97D9', '#6A5691']
plots_per_row = 2
num_rows = 2
fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(10, 6*num_rows))

for i, variable in enumerate(categorical_variables):
    category_counts = df_won[variable].value_counts()
    category_percentages = category_counts / category_counts.sum() * 100
    if num_rows > 1:
        ax = axes[i // plots_per_row, i % plots_per_row]
    else:
        ax = axes[i % plots_per_row]
    category_counts.plot(kind='bar', ax=ax, color=color[i])

    ax.set_xlabel(variable)
    ax.set_ylabel('Count')
```

```
    ax.set_title(f'{variable} Distribution')

    for j, count in enumerate(category_counts):
        percentage = category_percentages[j]
        ax.text(j, count + 2, f'{percentage:.1f}%', ha='center')
plt.tight_layout()
plt.show()
```


interior_seatsMaterial Distribution
69.3%
30.7%
Count
0
100
200
300
400
500
leather
cloth
interior_seatsMaterial


vehicle_lead_source Distribution
54.6%
44.7%
0.5%
0.1%
Count
0
100
200
300
400
500
600
700
DDC
MQL
FAMILY
AUTOZEN
vehicle_lead_source


general_transmission Distribution
94.2%
5.8%
Count
0
200
400
600
800
1000
1200
automatic
manual
general_transmission


general_roofType Distribution
98.3%
0.7%
0.6%
0.3%
0.1%
Count
0
200
400
600
800
1000
1200
hardTop
hardTopConvertible
softTopConvertible
softTop
softTopConverible
general_roofType

Visualizing Ordinal Variables

Visualizing ordinal variables is crucial in the exploratory data analysis (EDA) process as it allows us to gain insights into their distribution and understand their impact on the target variable. Ordinal variables have a specific order or ranking associated with their categories, making their visualization particularly informative. By examining the distribution of ordinal variables, we can assess the frequency and proportion of each category, identify any imbalances, and observe the patterns or trends present.

Understanding the distribution of ordinal variables helps us uncover potential relationships or associations between them and the target variable. By visualizing the stacked horizontal bar chart, we can observe the distribution of each ordinal variable's levels in a comprehensive and intuitive manner. This visualization enables us to identify dominant categories, spot outliers or unusual patterns, and gain a deeper understanding of the overall distribution of ordinal variables. Furthermore, examining the distribution can assist us in making informed decisions during feature engineering, such as identifying ordinal variables that may require encoding or transformation to enhance their predictive power.

```
ordinal_variables = df_won[AutozenFeatures.ordinal_feat_list]
ordinal_counts = ordinal_variables.apply(pd.Series.value_counts)
ordinal_counts = ordinal_counts.transpose()

fig = go.Figure()
for column in ordinal_counts.columns:
    # Add a horizontal bar trace for each level in the ordinal variable
    fig.add_trace(go.Bar(
        y=ordinal_counts.index,
        x=ordinal_counts[column],
        name=column,
        orientation='h'
    ))

fig.update_layout(
    barmode='stack',
    title='Stacked Horizontal Bar Chart - Ordinal Variables',
    xaxis_title='Count',
    yaxis_title='Ordinal Variables',
    height=1700,
    width=1200
)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Basic Statistics on the Autozen Data

In this section, we delve into the basic statistics of the Autozen dataset to gain an overview of the data. We explore various aspects, such as the most common car in the dataset, the distribution of cars by make, the most popular trim levels, the distribution of fuel types, cars with generally high mileage, cars with a generally high winning bid, and the time of the year with the highest bid.

- Determining the most popular trim levels provides valuable information on the preferred features or specifications that buyers seek in their vehicles. Understanding the distribution of fuel types helps us gauge the prevalence of different fuel sources, such as gasoline, diesel, hybrid, or electric, which can be indicative of environmental concerns or evolving technology.
- Investigating cars with generally high mileage helps identify potential outliers or instances of heavy usage. Similarly, examining cars with a generally high winning bid sheds light on valuable or sought-after vehicles. These analyses can uncover patterns or factors that contribute to higher prices or indicate specific market segments.
- Examining the time of the year with the highest bid allows us to identify potential seasonal variations in demand or bidding behavior, providing insights into the market dynamics and potential influencing factors.

Most common car make that occurs in the data?

By analyzing the most common car in the data, we can identify the dominant car model or brand within the dataset, providing insights into consumer preferences or market trends. Additionally, examining the number of cars by make allows us to understand the distribution and representation of different car manufacturers in the dataset. From the bar graph below, we observe that we have Toyota as the most common car in the dataset forming 9.3% of the total and 14.1% of the top 10 Car Makes.

```
car_make_counts = df_won['general_make'].value_counts(
).sort_values(ascending=True)
car_make_counts_df = car_make_counts.reset_index()
car_make_counts_df.columns = ['Car Make', 'Number of Cars']
total_cars = car_make_counts.sum()
car_make_counts_df['Percentage'] = car_make_counts_df['Number of Cars'] / \
    total_cars * 100


fig = sp.make_subplots(rows=1, cols=2, subplot_titles=('Most Common Car Make in the Entire
```

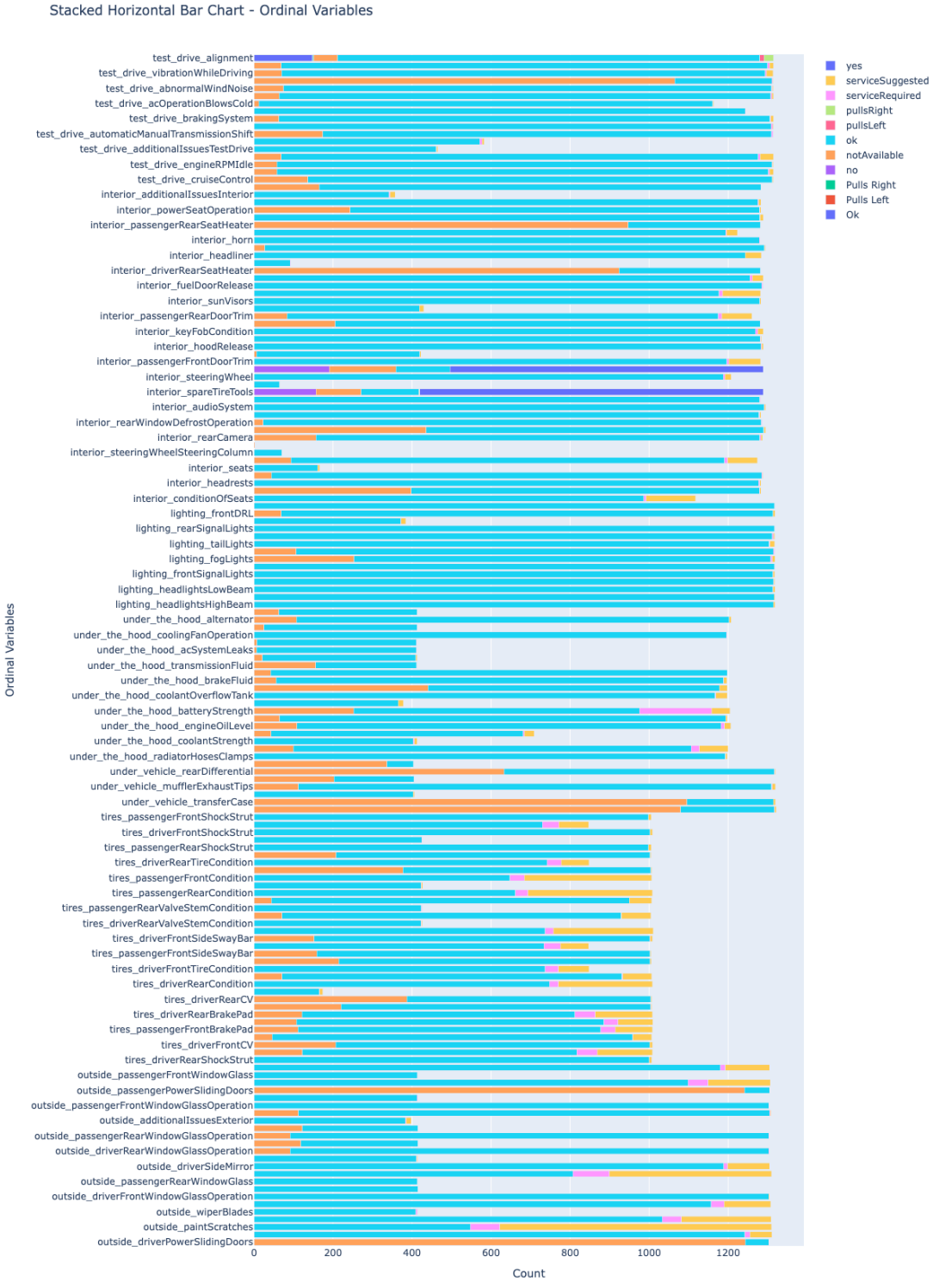


Figure 2: ordinalbar

```
                        specs=[[{"type": "bar"}, {"type": "pie"}]])
fig.add_trace(go.Bar(x=car_make_counts_df['Number of Cars'], y=car_make_counts_df['Car Mak
                     orientation='h', text=car_make_counts_df['Percentage'].map('{:.2f}%'.
                     textposition='outside'),
              row=1, col=1)
fig.add_trace(go.Pie(labels=car_make_counts_df['Car Make'][-10:],
                     values=car_make_counts_df['Number of Cars'][-10:], textinfo='label+pe
fig.update_layout(showlegend=False, title_text='Number of Cars Won by Make')

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

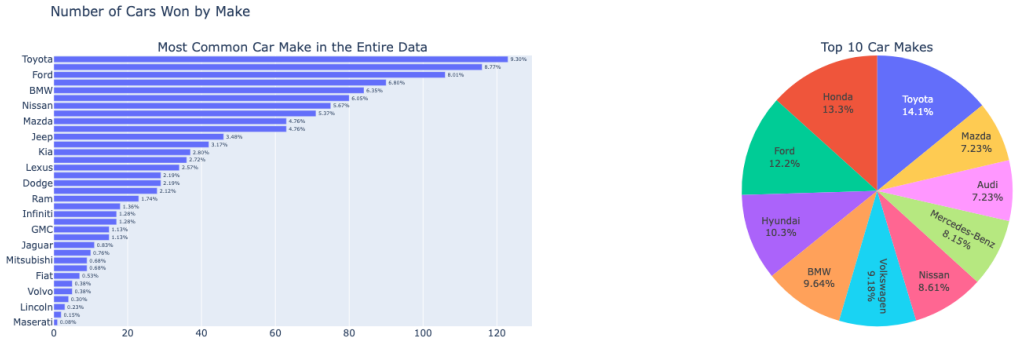
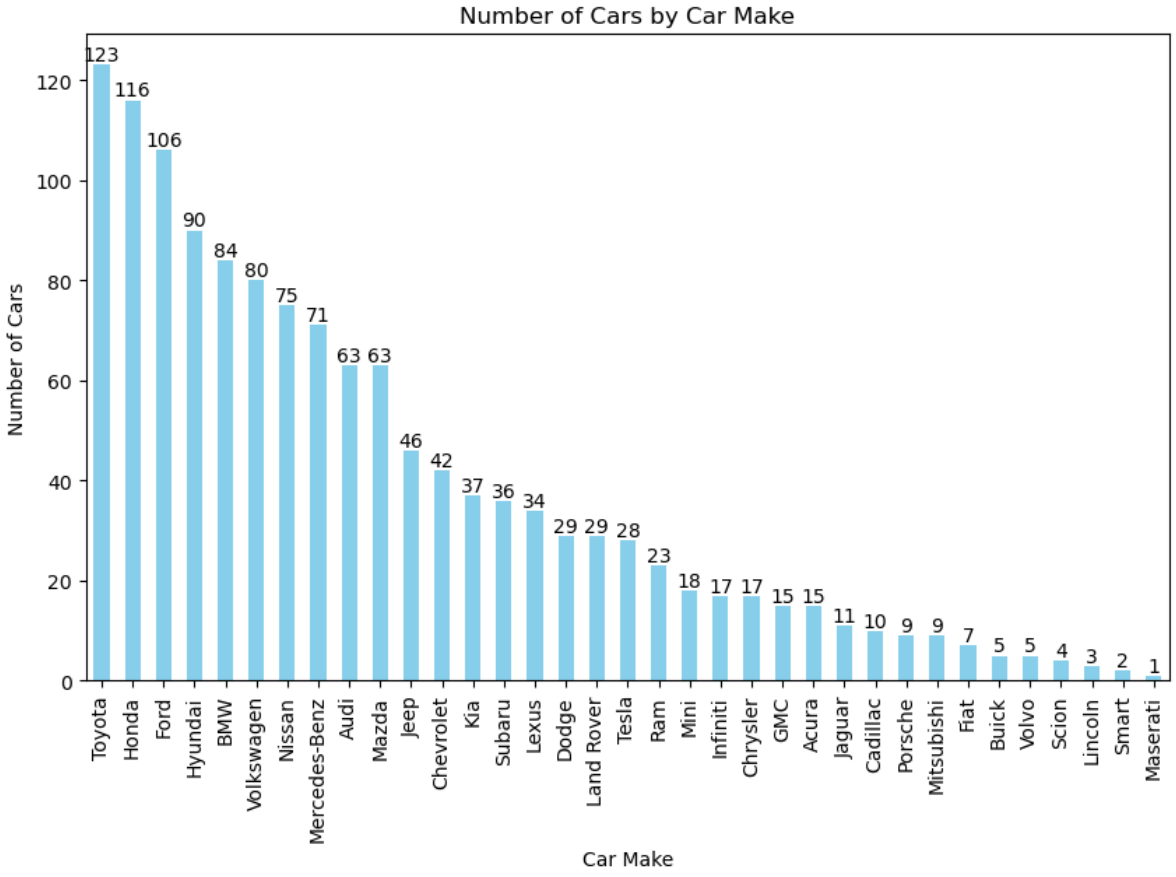


Figure 3: makebar

```
car_make_counts = df_won['general_make'].value_counts().sort_values(ascending=False)
plt.figure(figsize=(10, 6))
car_make_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Car Make')
plt.ylabel('Number of Cars')
plt.title('Number of Cars by Car Make')
for i, count in enumerate(car_make_counts):
    plt.text(i, count, str(count), ha='center', va='bottom')
plt.show()
```

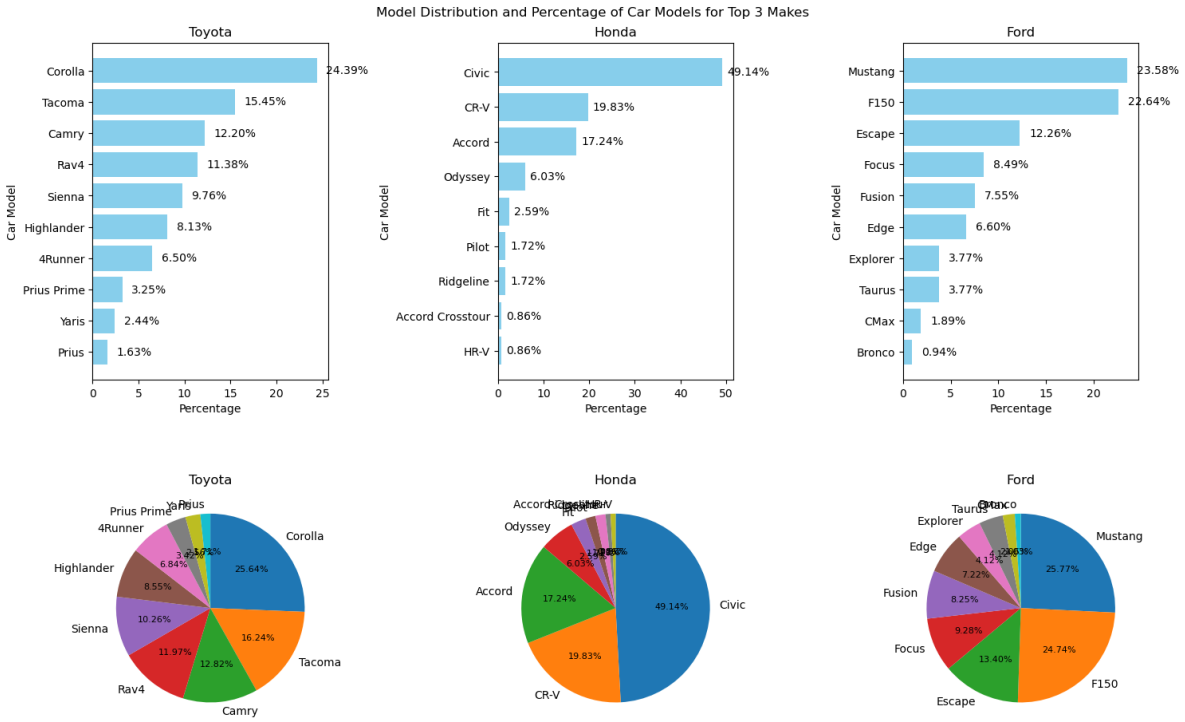


Highest Car Models in the Top 3 Car Make Brands?

Exploring the distribution and percentage of car models for the top three car make brands. The data is obtained by filtering the dataset to include only the top three car make brands based on their frequency. The code then calculates the percentage of each car model within each car make brand. The resulting data is sorted by car make and percentage in descending order.

The findings show the highest car models in the top three car make brands. For the Ford brand, the Mustang and F150 are the most prominent models, accounting for 23.58% and 22.64% respectively. In the Honda brand, the Civic is the dominant model, representing 49.14% of the total. For the Toyota brand, the Corolla is the leading model, comprising 24.39% of the total. The bar graphs and pie charts visualize the distribution and percentages of the top car models for each car make brand, providing a clear comparison of the model preferences within the top three brands.

```
top_3_makes = df_won['general_make'].value_counts().nlargest(3).index.tolist()
df_top_3_makes = df_won[df_won['general_make'].isin(top_3_makes)]
model_percentages = (df_top_3_makes.groupby(['general_make', 'general_model'])
                     .size() / df_top_3_makes.groupby('general_make').size() * 100)
model_percentages = model_percentages.reset_index()
model_percentages.columns = ['Car Make', 'Car Model', 'Percentage']
model_percentages.sort_values(by=['Car Make', 'Percentage'], ascending=[
                              True, False], inplace=True)

num_makes = len(top_3_makes)
fig, axes = plt.subplots(2, num_makes, figsize=(15, 10))
fig.suptitle('Model Distribution and Percentage of Car Models for Top 3 Makes')

for i, make in enumerate(top_3_makes):
    make_data = model_percentages[model_percentages['Car Make'] == make]
    percentages = make_data['Percentage'].head(10)
    models = make_data['Car Model'].head(10)

    ax1 = axes[0, i]
    ax1.barh(models[::-1], percentages[::-1], color='skyblue')
    ax1.set_xlabel('Percentage')
    ax1.set_ylabel('Car Model')
    ax1.set_title(make)

    for j, (percentage, model) in enumerate(zip(percentages[::-1], models[::-1])):
        ax1.text(percentage + 1, j,
                 f'{percentage:.2f}%', ha='left', va='center')
    ax2 = axes[1, i]
    pie_wedges, _, text_labels = ax2.pie(
        percentages, labels=models, autopct='%.2f%%', startangle=90, counterclock=False)
    for label in text_labels:
        label.set_fontsize(8)
    ax2.set_title(make)

plt.tight_layout()
plt.show()
```




```
num_makes = len(top_3_makes)
fig = sp.make_subplots(rows=2, cols=num_makes, subplot_titles=top_3_makes, specs=[[{'type'

for i, make in enumerate(top_3_makes):
    make_data = model_percentages[model_percentages['Car Make'] == make]
    percentages = make_data['Percentage'].head(10)
    models = make_data['Car Model'].head(10)

    fig.add_trace(
        go.Bar(x=percentages[::-1], y=models[::-1], orientation='h', marker_color='skyblue
        row=1, col=i+1)

    fig.add_trace(
        go.Pie(labels=models, values=percentages, textinfo='label+percent', hole=0.4),
        row=2, col=i+1)

fig.update_layout(
    title='Model Distribution and Percentage of Car Models for Top 3 Makes',
    height=600,
    width=1200,
```

```
    showlegend=False
)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

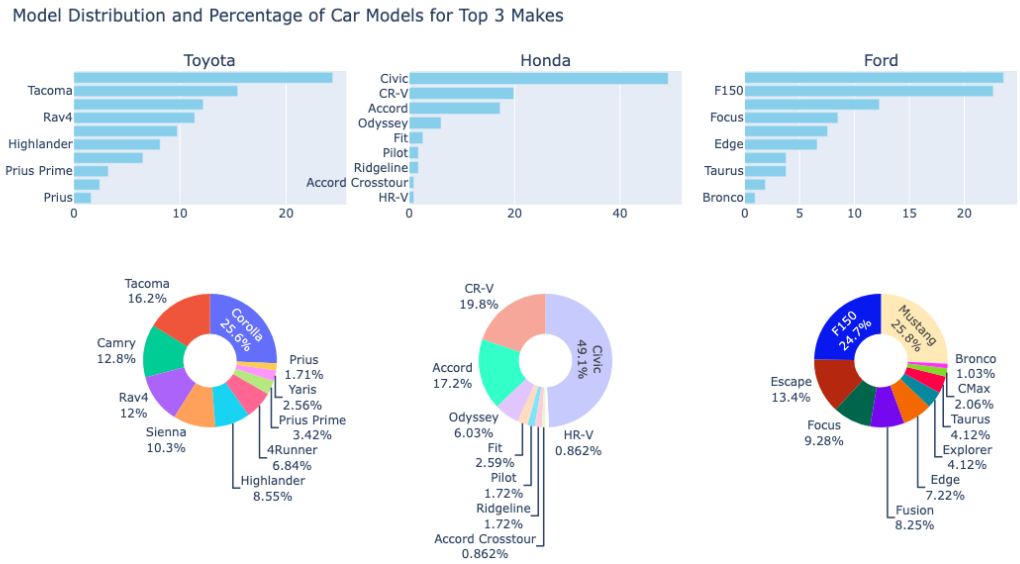


Figure 4: modelbar

Fuel Type Dsitribution

The fuel types analyzed are electric, diesel, hybrid, and gasoline, with a grand total of 1,143 vehicles as seen on the cross tab below. The findings from the analysis indicate that electric vehicles have the highest average bid winning price at $50,007, followed by diesel vehicles at $31,124, hybrid vehicles at $26,702, and gasoline vehicles at $24,559. Similarly, the average values for CBB high and AZ high are highest for electric vehicles, while CBB low and AZ low values are highest for diesel vehicles.

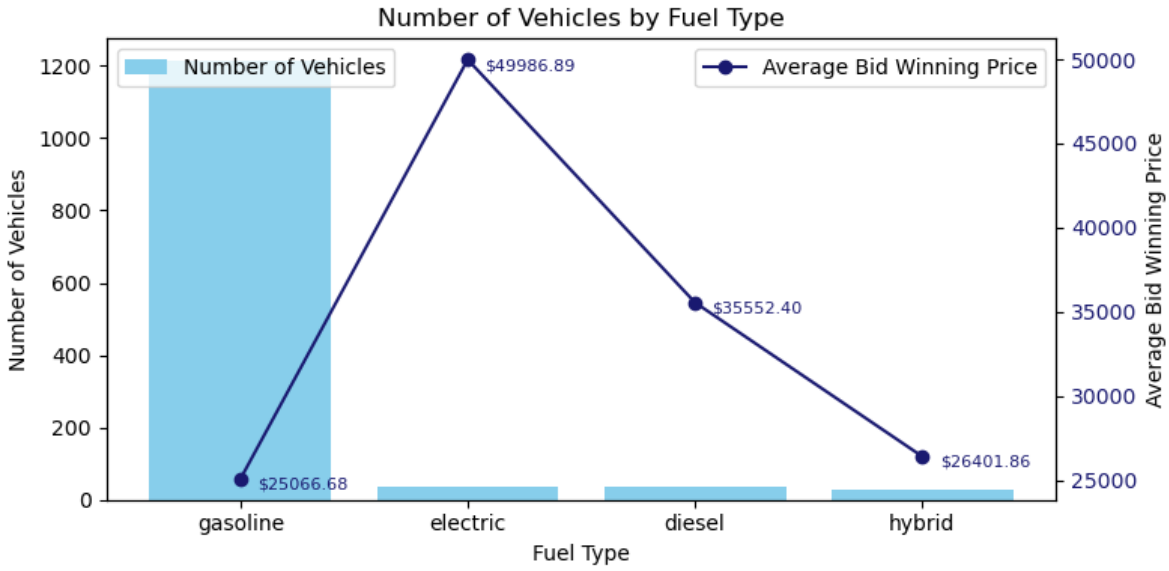
```
df_grouped = df_won.groupby('general_fuelType').agg({
    'general_fuelType': 'count',
    'valuation_highest_bid_amount': 'mean'
}).rename(columns={'general_fuelType': 'Count',
                   'valuation_highest_bid_amount': 'Average Bid Winning Price'}).sort_valu
```

Fuel Breakdown with Pricing Comaprison

general fuelType	Number of Vehicles	Avg. Bid Start	Avg. Bid Winning	Avg. CBB High	Avg. AZ High	Avg. CBB Low	Avg. AZ Low
electric	38	49,229	50,007	53,357	55,936	48,866	48,192
diesel	28	30,746	31,124	30,869	31,577	27,580	28,073
hybrid	28	25,304	26,702	26,830	28,371	24,247	24,779
gasoline	1,046	23,727	24,559	24,118	25,675	21,698	23,113
Grand Total	**1,143**	**24,825**	**25,656**	**25,376**	**26,930**	**22,858**	**24,140**

Figure 5: FuelComp

```
fig, ax1 = plt.subplots(figsize=(8, 4))
ax1.bar(df_grouped.index, df_grouped['Count'], color='skyblue')
ax1.set_xlabel('Fuel Type')
ax1.set_ylabel('Number of Vehicles')
ax1.set_title('Number of Vehicles by Fuel Type')

ax2 = ax1.twinx()
ax2.plot(df_grouped.index,
         df_grouped['Average Bid Winning Price'], color='midnightblue', marker='o')
ax2.set_ylabel('Average Bid Winning Price')
ax2.tick_params(axis='y', labelcolor='midnightblue')

for i, val in enumerate(df_grouped['Average Bid Winning Price']):
    ax2.annotate(f"${val:.2f}", (i, val), xytext=(30, -5), textcoords='offset points',
                 ha='center', fontsize=8, color='midnightblue')

plt.xticks(rotation=45)
ax1.legend(['Number of Vehicles'], loc='upper left')
ax2.legend(['Average Bid Winning Price'], loc='upper right')
plt.tight_layout()
plt.show()
```



Cars with Highest Mileage and Highest Winning Bid

Here we analyze the average bid winning prices and general mileage for various car makes. The table presents the top car makes along with their corresponding average bid winning prices and average general mileage.

Summary of Findings:

- The car make with the highest average bid winning price is Porsche, with an average of $66,295.78.
- The car make with the highest average general mileage is Lincoln, with an average of 105,135 miles.
- Some car makes, such as Smart and Maserati, have relatively low average bid winning prices but also low general mileage.
- On the other hand, car makes like Tesla and Porsche have higher average bid winning prices but lower general mileage compared to other models.
- Overall, there is a varied range of average bid winning prices and general mileage across different car makes, indicating a diverse market for both high-end and more affordable vehicles.

```
df_winning_bids = df_won.groupby('general_make').agg({
    'valuation_highest_bid_amount': 'mean',
    'general_mileage': 'mean'
}).reset_index()
```

```
df_winning_bids = df_winning_bids.sort_values('valuation_highest_bid_amount', ascending=Fa
fig1 = go.Figure(data=go.Bar(
    x=df_winning_bids['general_make'],
    y=df_winning_bids['valuation_highest_bid_amount'],
    marker=dict(
        color=df_winning_bids['general_mileage'],
        colorscale='Blues',
        colorbar=dict(title='Average Mileage')
    ),
    hovertemplate='<b>%{x}</b><br>'
                  'Average Winning Bid: $%{y:.2f}<br>'
                  'Average Mileage: %{y:.2f}'
))

for x, y in zip(df_winning_bids['general_make'], df_winning_bids['valuation_highest_bid_am
    fig1.add_annotation(
        x=x, y=y,
        text=f'${y:.2f}',
        showarrow=False,
        font=dict(size=8),
        xanchor='center',
        yanchor='bottom',
        textangle=90
    )

fig1.update_layout(
    title='Car Makes with the Highest Winning Bids',
    xaxis_title='Car Make',
    yaxis_title='Average Highest Bid Amount'
)
fig1.show()
```

```
Unable to display output for mime type(s): text/html
```

Time of the Year with the Highest Bids

Here, we investigate the time of the year when the average winning bids are the highest. This investigation is crucial in understanding the seasonal patterns and trends in the bidding activity. By examining the data, we can identify the months that tend to yield higher average winning bids, providing valuable insights for auction strategy and decision-making.

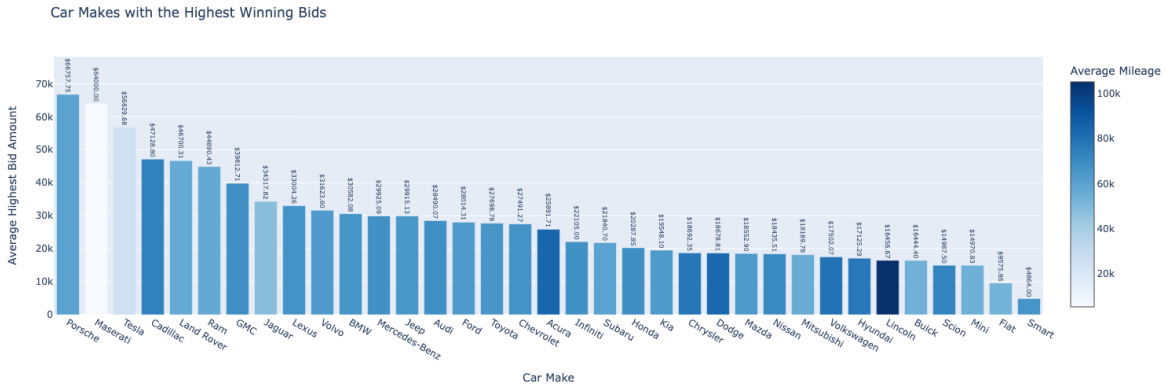


Figure 6: milbid

The results show that the average winning bids vary across different months of the year. The summary of the findings is as follows:

- January: $30,302.90
- February: $25,736.19
- March: $27,558.30
- April: $24,351.63
- May: $18,967.85
- June: $19,494.50
- July: $20,830.14
- August: $24,317.99
- September: $23,918.59
- October: $24,816.96
- November: $26,867.38
- December: $30,512.68

The analysis reveals that the months of January and December have the highest average winning bids, indicating a potential seasonality effect during the winter months. On the other hand, May and June have relatively lower average winning bids. These insights can assist in strategic planning and decision-making when participating in auctions during specific times of the year.

```
import calendar

df_monthly_avg_bids = df_won.groupby('month_of_year')['valuation_highest_bid_amount'].mean
df_monthly_avg_bids = df_monthly_avg_bids.sort_values('month_of_year')

fig = go.Figure(data=go.Scatter(x=df_monthly_avg_bids['month_of_year'],
```

```
                                y=df_monthly_avg_bids['valuation_highest_bid_amount'],
                                mode='lines',
                                fill='tozeroy',
                                line=dict(color='midnightblue'),
                                hovertemplate='Month: %{customdata}<br>Avg. Highest Bid: $%
                                customdata=[calendar.month_name[m] for m in df_monthly_avg_
                                text=[f"${val:.2f}" for val in df_monthly_avg_bids['valuati
                                textposition="top center",
                                textfont=dict(color='midnightblue', size=8)))

fig.update_traces(mode='lines+markers', marker=dict(symbol='circle', size=5))

fig.update_layout(title='Time of the year with the highest Bids',
                  xaxis_title='Month',
                  yaxis_title='Average Highest Bid Amount',
                  xaxis=dict(
                      tickmode='array',
                      tickvals=df_monthly_avg_bids['month_of_year'],
                      ticktext=[calendar.month_name[m] for m in df_monthly_avg_bids['month
                  ))

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

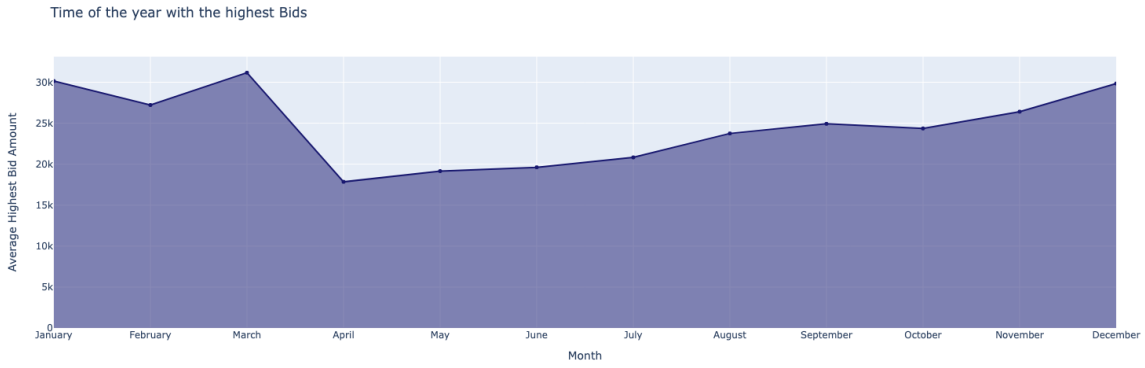


Figure 7: timebid

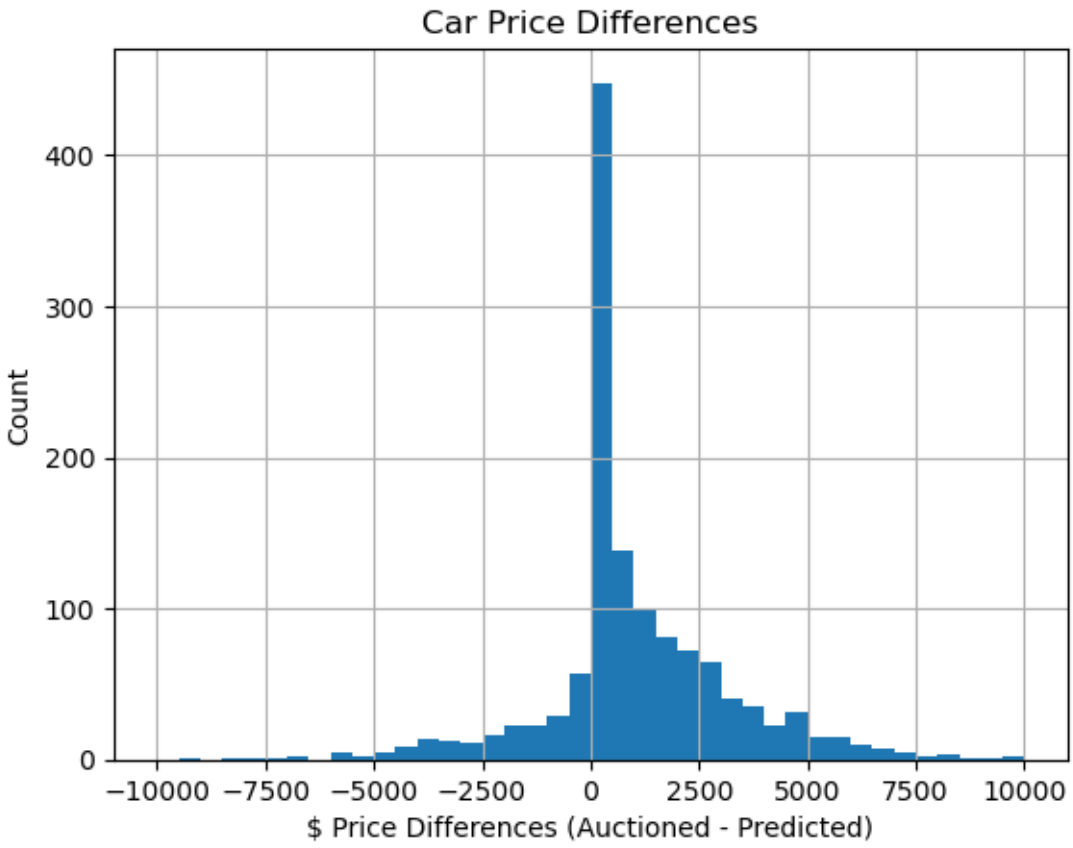
By examining these basic statistics, we gain a comprehensive understanding of the Autozen dataset, enabling us to make informed decisions, generate hypotheses for further exploration, or identify potential areas of interest for subsequent analysis.

Price distribution of sold used cars

Distribution of difference in price between Autozen valuation and winning bids

The provided code calculates the difference in price between the winning bids and the starting prices for vehicles in the filtered and merged auctions dataset. It then creates a histogram to visualize the distribution of these price differences. We need to look at this code to understand the distribution of the price differences between the auctioned prices and the predicted prices by Autozen. This analysis helps us assess the accuracy and effectiveness of the Autozen valuation in predicting the final auction prices. The histogram provides insights into the range and frequency of the price differences, allowing us to evaluate any potential biases or discrepancies in the valuation process.

```
values_df = df_won[['valuation_starting_price', 'bid_winning']].dropna(axis=0)
values_df['difference'] = values_df['bid_winning'] - values_df['valuation_starting_price']
# Plot a histogram of the bid_winning column
values_df['difference'].hist(bins=40, range=(-10000, 10000))

# Add axis labels and a title
plt.xlabel('$ Price Differences (Auctioned - Predicted)')
plt.ylabel('Count')
plt.title('Car Price Differences')

# Show the plot
plt.show()
```



CBB and Autozen valuations relative to Auction Price

This code performs evaluation and comparison between predictions made by Autozen and CBB (Car Book Value) models for vehicle valuation. It calculates various evaluation metrics such as R-squared (R2), Mean Absolute Percentage Error (MAPE), Negative Mean Squared Error (MSE), and Negative Root Mean Squared Error (RMSE). The results are stored in a DataFrame called "Autozen_CBB_evaluation_all_time" for further analysis and comparison.

```
# Autozen features are defined in AutozenFeatures in the scripts modules
MERGED_PROCESSED_pred = os.path.join(data_dir,"predicted_results.csv")
pred_df = pd.read_csv(MERGED_PROCESSED_pred)
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
pred_df = pred_df[
    [
```

```
        "valuation_cbb_trade_in_low",
        "valuation_cbb_trade_in_high",
        "valuation_autozen_low",
        "valuation_autozen_high",
        "Predicted_75_lower_bound",
        "Predicted_75_upper_bound",
        "Predicted_price",
        "created_at"
    ]
]
pred_df['bid_winning'] = df_won['bid_winning']

# Convert "created_at" to datetime
pred_df["created_at"] = pd.to_datetime(pred_df["created_at"])

# Group by quarter year
pred_df["quarter_year"] = pred_df["created_at"].dt.to_period("Q")


# Replace null and NaN values in valuation_cbb_trade_in_low with values from valuation_cbb
pred_df["valuation_cbb_trade_in_low"] = pred_df["valuation_cbb_trade_in_low"].fillna(
    pred_df["valuation_cbb_trade_in_high"]
)
pred_df["valuation_cbb_trade_in_high"] = pred_df["valuation_cbb_trade_in_high"].fillna(
    pred_df["valuation_cbb_trade_in_low"]
)

# Calculate the mean value for each pair of valuation_cbb_trade_in_low and valuation_cbb_t
pred_df["valuation_cbb_trade_in_mean"] = np.average(
    pred_df[["valuation_cbb_trade_in_low", "valuation_cbb_trade_in_high"]], axis=1
)

# Calculate the mean value for each pair of valuation_autozen_low and valuation_autozen_hi
pred_df["valuation_autozen_mean"] = np.average(
    pred_df[["valuation_autozen_low", "valuation_autozen_high"]], axis=1
)

cbb_nonnull_pred_df = pred_df[
    pred_df[["valuation_cbb_trade_in_mean", "bid_winning"]].notnull().all(axis=1)
]
```

```
cbb_preds = cbb_nonnull_pred_df["valuation_cbb_trade_in_mean"]
az_preds = pred_df["valuation_autozen_mean"]
capstone_preds = pred_df['Predicted_price']
y_cbb = cbb_nonnull_pred_df["bid_winning"]
y_az = pred_df["bid_winning"]
y_capstone = pred_df["bid_winning"]

def evaluation(y, preds, quarter, pre_name):
    scoring = {}
    scoring["Prediction"] = pre_name
    scoring['Quarter'] = quarter
    scoring['Data Count'] = len(y)
    scoring["R2"] = r2_score(y, preds)
    scoring["MAPE"] = mean_absolute_percentage_error(y, preds) * 100
    scoring["Negative MSE"] = -mean_squared_error(y, preds)
    scoring["Negative RMSE"] = -np.sqrt(mean_squared_error(y, preds))
    return scoring


scoring_list = []
scoring_list.append(evaluation(y_cbb, cbb_preds, "All-time", "CBB"))
scoring_list.append(evaluation(y_az, az_preds,"All-time", "Autozen"))
scoring_list.append(evaluation(y_capstone, capstone_preds,"All-time", "Capstone"))

Autozen_CBB_evaluation_all_time = pd.DataFrame(scoring_list)
Autozen_CBB_evaluation_all_time
```

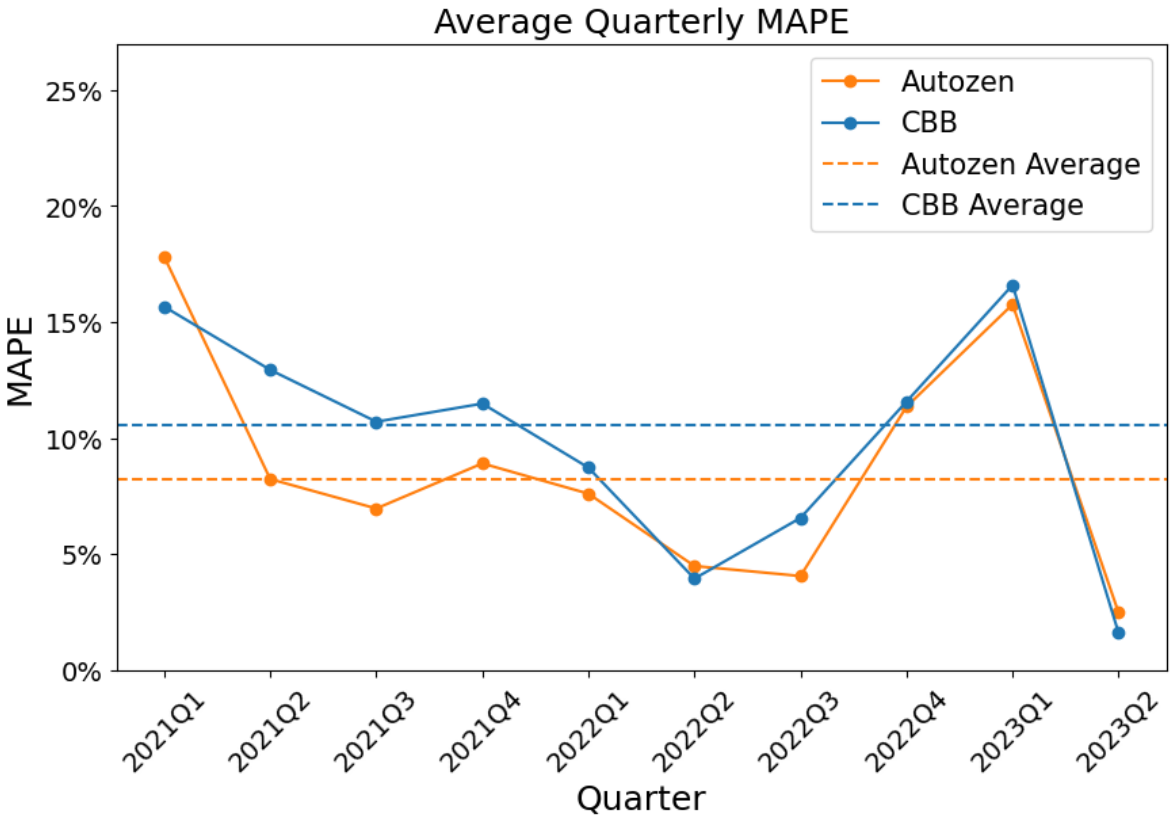
	Prediction	Quarter	Data Count	R2	MAPE	Negative MSE	Negative RMSE
0	CBB	All-time	1046	-1.349249	56.989361	-5.398044e+08	-23233.690074
1	Autozen	All-time	1058	-1.370216	58.145023	-5.417407e+08	-23275.324042
2	Capstone	All-time	1058	-1.287251	57.272564	-5.227782e+08	-22864.343486

How that difference varied over the lifetime of the dataset ?

```
import matplotlib.ticker as mtick
def evaluation(y, preds, quarter, pre_name):
    scoring = {}
    scoring["Prediction"] = pre_name
    scoring['Quarter'] = quarter
```

```
    scoring["MAPE"] = mean_absolute_percentage_error(y, preds) * 100
    scoring["Data Count"] = len(y)
    return scoring

scoring_list = []

# Group by quarter year and apply the evaluation metric to each group
for quarter, group in pred_df.groupby("quarter_year"):
    y_az_quarter = group["bid_winning"]
    az_preds_quarter = group["valuation_autozen_mean"]
    scoring_list.append(evaluation(y_az_quarter, az_preds_quarter, quarter, "Autozen"))

for quarter, group in cbb_nonnull_pred_df.groupby("quarter_year"):
    y_cbb_quarter = group["bid_winning"]
    cbb_preds_quarter = group["valuation_cbb_trade_in_mean"]
    scoring_list.append(evaluation(y_cbb_quarter, cbb_preds_quarter, quarter, "CBB"))

# for quarter, group in pred_df.groupby("quarter_year"):
#     y_capstone_quarter = group["bid_winning"]
#     capstone_preds_quarter = group["Predicted_price"]
#     scoring_list.append(evaluation(y_az_quarter, az_preds_quarter, quarter, "Capstone"))

colors = {
    'CBB': 'C0',
    'Autozen': 'C1'
}

Autozen_CBB_evaluation = pd.DataFrame(scoring_list)
fig, ax = plt.subplots(figsize=(10, 6))
for prediction, group in Autozen_CBB_evaluation.groupby("Prediction"):
    ax.plot(group["Quarter"].astype(str), group["MAPE"], marker='o', color=colors[predicti

# Calculate the average MAPE
average_mape = Autozen_CBB_evaluation["MAPE"].mean()

ax.axhline(Autozen_CBB_evaluation_all_time['MAPE'][1], color='C1', linestyle='--', label='
ax.axhline(Autozen_CBB_evaluation_all_time['MAPE'][0], color='C0', linestyle='--', label='
# ax.axhline(0.034557 * 100, color='C3', linestyle='--', label='XGBoost')
# ax.axhline(0.029926 * 100, color='C4', linestyle='--', label='XGBoost + Lasso Optimized'
# ax.axhline(0.09 * 100, color='C5', linestyle='--', label='Neural Network')
ax.set_xlabel("Quarter", fontsize=18)
```

```
ax.set_ylabel("MAPE", fontsize=18)
ax.set_title("Average Quarterly MAPE", fontsize=18)

fmt = '%.0f%%'
yticks = mtick.FormatStrFormatter(fmt)
ax.yaxis.set_major_formatter(yticks)
ax.tick_params(axis='both', labelsize=14)

# Add legend with increased font size
plt.legend(fontsize=15)
plt.xticks(rotation=45)
# plt.grid(True)
plt.ylim(0, 27)
plt.show()
Autozen_CBB_evaluation
```



	Prediction	Quarter	MAPE	Data Count
0	Autozen	2021Q1	17.810584	50
1	Autozen	2021Q2	8.228249	175
2	Autozen	2021Q3	6.959611	270
3	Autozen	2021Q4	8.903736	316
4	Autozen	2022Q1	7.599789	493
5	Autozen	2022Q2	4.481478	5
6	Autozen	2022Q3	4.049440	7
7	Autozen	2022Q4	11.351662	3
8	Autozen	2023Q1	15.759541	3
9	Autozen	2023Q2	2.466531	1
10	CBB	2021Q1	15.662661	50
11	CBB	2021Q2	12.935618	158
12	CBB	2021Q3	10.701924	270
13	CBB	2021Q4	11.483345	316
14	CBB	2022Q1	8.723249	493
15	CBB	2022Q2	3.937450	5
16	CBB	2022Q3	6.554721	7
17	CBB	2022Q4	11.570473	3
18	CBB	2023Q1	16.594031	3
19	CBB	2023Q2	1.590264	1

Was the Auction Price ouside or inside the Autozen and CBB Range ?

```
# Create a new column in values_df to indicate the bid_winning status
pred_df["bid_winning_status_cbb"] = None

# Set bid_winning_status to "within range" if bid_winning is within the low and high bound
pred_df.loc[
    (pred_df["bid_winning"] >= pred_df["valuation_cbb_trade_in_low"])
    & (pred_df["bid_winning"] <= pred_df["valuation_cbb_trade_in_high"]),
    "bid_winning_status_cbb",
] = "within range"

# Set bid_winning_status to "below range" if bid_winning is below the low bound
pred_df.loc[
    pred_df["bid_winning"] < pred_df["valuation_cbb_trade_in_low"],
    "bid_winning_status_cbb",
] = "below range"
```

```
# Set bid_winning_status to "above range" if bid_winning is above the high bound
pred_df.loc[
    pred_df["bid_winning"] > pred_df["valuation_cbb_trade_in_high"],
    "bid_winning_status_cbb",
] = "above range"

print(pred_df["bid_winning_status_cbb"].value_counts(normalize=True))
print(pred_df["bid_winning_status_cbb"].value_counts(normalize=False))



# Create a new column in values_df to indicate the bid_winning status
pred_df["bid_winning_status_az"] = None

# Set bid_winning_status to "within range" if bid_winning is within the low and high bound
pred_df.loc[
    (pred_df["bid_winning"] >= pred_df["valuation_autozen_low"])
    & (pred_df["bid_winning"] <= pred_df["valuation_autozen_high"]),
    "bid_winning_status_az",
] = "within range"

# Set bid_winning_status to "below range" if bid_winning is below the low bound
pred_df.loc[
    pred_df["bid_winning"] < pred_df["valuation_autozen_low"], "bid_winning_status_az"
] = "below range"

# Set bid_winning_status to "above range" if bid_winning is above the high bound
pred_df.loc[
    pred_df["bid_winning"] > pred_df["valuation_autozen_high"], "bid_winning_status_az"
] = "above range"

print(pred_df["bid_winning_status_az"].value_counts(normalize=True))
print(pred_df["bid_winning_status_az"].value_counts(normalize=False))



# Create a new column in values_df to indicate the bid_winning status
pred_df["bid_winning_status_capstone"] = None

# Set bid_winning_status to "within range" if bid_winning is within the low and high bound
pred_df.loc[
```

```
    (pred_df["bid_winning"] >= pred_df["Predicted_75_lower_bound"])
    & (pred_df["bid_winning"] <= pred_df["Predicted_75_upper_bound"]),
    "bid_winning_status_capstone",
] = "within range"

# Set bid_winning_status to "below range" if bid_winning is below the low bound
pred_df.loc[
    pred_df["bid_winning"] < pred_df["Predicted_75_lower_bound"], "bid_winning_status_caps
] = "below range"

# Set bid_winning_status to "above range" if bid_winning is above the high bound
pred_df.loc[
    pred_df["bid_winning"] > pred_df["Predicted_75_upper_bound"], "bid_winning_status_caps
] = "above range"

print(pred_df["bid_winning_status_capstone"].value_counts(normalize=True))
print(pred_df["bid_winning_status_capstone"].value_counts(normalize=False))


# Calculate value counts and normalize the result for 'bid_winning_status_cbb'
status_cbb_normalized = (
    pred_df["bid_winning_status_cbb"].value_counts(normalize=True).reset_index()
)
status_cbb_normalized.columns = ["bid_winning_status_cbb", "normalized_count"]

status_cbb_unnormalized = (
    pred_df["bid_winning_status_cbb"].value_counts(normalize=False).reset_index()
)
status_cbb_unnormalized.columns = ["bid_winning_status_cbb", "count"]

# Calculate value counts and normalize the result for 'bid_winning_status_az'
status_az_normalized = (
    pred_df["bid_winning_status_az"].value_counts(normalize=True).reset_index()
)
status_az_normalized.columns = ["bid_winning_status_az", "normalized_count"]

status_az_unnormalized = (
    pred_df["bid_winning_status_az"].value_counts(normalize=False).reset_index()
)
status_az_unnormalized.columns = ["bid_winning_status_az", "count"]
```

```
# Calculate value counts and normalize the result for 'bid_winning_status_capstone'
status_capstone_normalized = (
    pred_df["bid_winning_status_capstone"].value_counts(normalize=True).reset_index()
)
status_capstone_normalized.columns = ["bid_winning_status_capstone", "normalized_count"]

status_capstone_unnormalized = (
    pred_df["bid_winning_status_capstone"].value_counts(normalize=False).reset_index()
)
status_capstone_unnormalized.columns = ["bid_winning_status_capstone", "count"]

# Create a DataFrame to store the results
bid_winning_status_counts = pd.DataFrame(
    {
        "index": status_cbb_unnormalized["bid_winning_status_cbb"],
        "CBB": status_cbb_unnormalized["count"],
        "Autozen": status_az_unnormalized["count"],
        "Capstone": status_capstone_unnormalized["count"],
    }
)

# Print the result DataFrame
bid_winning_status_counts

bid_winning_status_counts_melt = bid_winning_status_counts.melt(
    value_vars=["CBB", "Autozen", "Capstone"], id_vars="index"
)
bid_winning_status_counts_melt
```

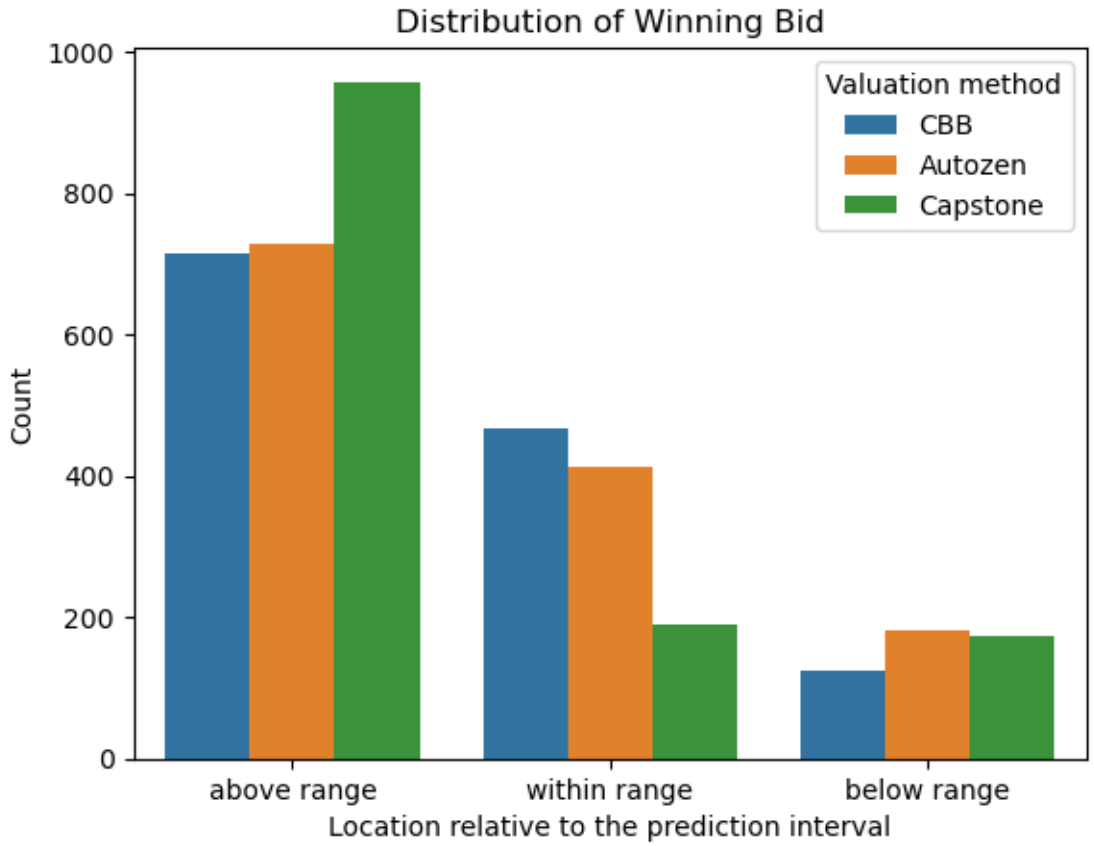
```
bid_winning_status_cbb
above range     0.547473
within range    0.357580
below range     0.094946
Name: proportion, dtype: float64
bid_winning_status_cbb
above range     715
within range    467
below range     124
Name: count, dtype: int64
bid_winning_status_az
within range    0.550265
above range     0.312169
```

```
below range     0.137566
Name: proportion, dtype: float64
bid_winning_status_az
within range    728
above range     413
below range     182
Name: count, dtype: int64
bid_winning_status_capstone
within range    0.724112
above range     0.144369
below range     0.131519
Name: proportion, dtype: float64
bid_winning_status_capstone
within range    958
above range     191
below range     174
Name: count, dtype: int64
```

	index	variable	value
0	above range	CBB	715
1	within range	CBB	467
2	below range	CBB	124
3	above range	Autozen	728
4	within range	Autozen	413
5	below range	Autozen	182
6	above range	Capstone	958
7	within range	Capstone	191
8	below range	Capstone	174

```
# Plot as a bar chart with variable as color
sns.barplot(
    data=bid_winning_status_counts_melt,
    x="index",
    y="value",
    hue="variable",
)

# Set the plot labels and title
plt.xlabel("Location relative to the prediction interval")
plt.ylabel("Count")
```

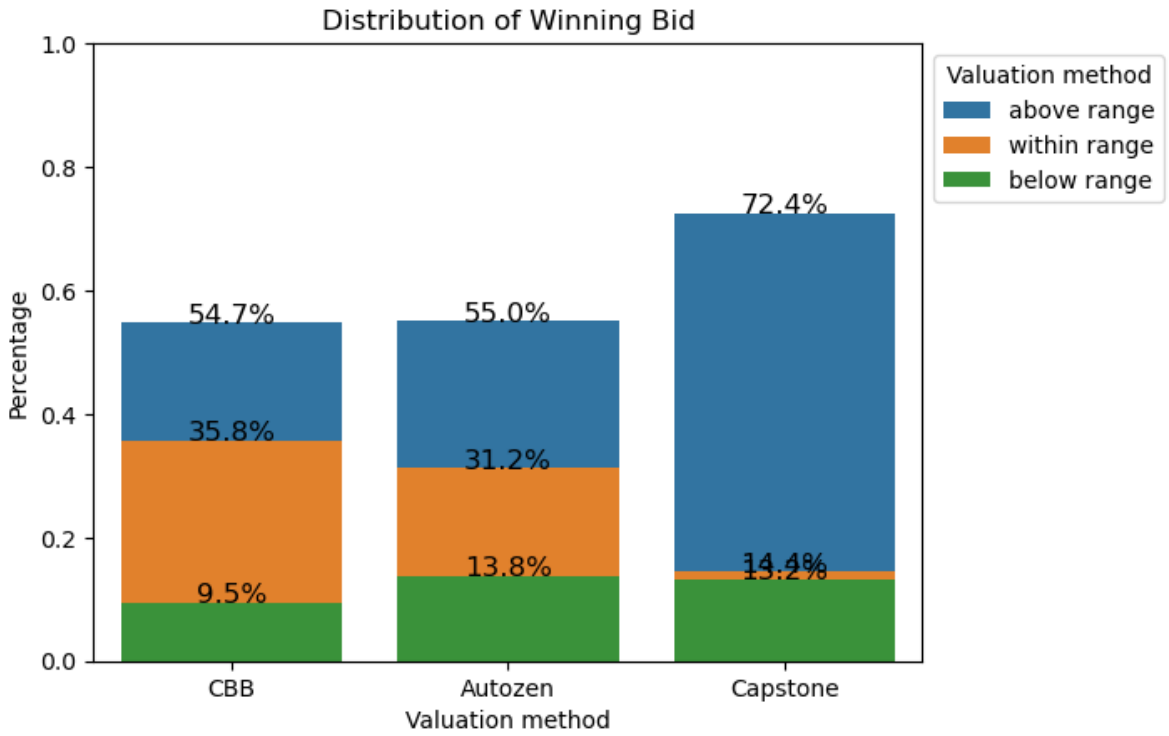
```
plt.title("Distribution of Winning Bid")

# Modify legend labels
plt.legend(title="Valuation method")

# Show the plot
plt.show()
```




```
# Create a DataFrame to store the results
bid_winning_status_norm = pd.DataFrame(
    {
        "index": status_cbb_normalized["bid_winning_status_cbb"],
        "CBB": status_cbb_normalized["normalized_count"],
        "Autozen": status_az_normalized["normalized_count"],
        "Capstone": status_capstone_normalized["normalized_count"],
```

```
    }
)

# Melt the DataFrame for plotting
bid_winning_status_norm_melt = bid_winning_status_norm.melt(
    value_vars=["CBB", "Autozen", "Capstone"], id_vars="index"
)

# Plot as a stacked bar chart
barplot = sns.barplot(
    data=bid_winning_status_norm_melt,
    x="variable",
    y="value",
    hue="index",
    dodge=False,
    linewidth=0,
    edgecolor="none",
)

# Set the plot labels and title
plt.xlabel("Valuation method")
plt.ylabel("Percentage")
plt.title("Distribution of Winning Bid")

# Modify legend labels
plt.legend(title="Valuation method", bbox_to_anchor=(1, 1), loc="upper left")

# Add data labels
for p in barplot.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    barplot.text(
        x=x + width / 2,
        y=y + height,
        s=f"{height:.1%}",
        ha="center",
        color="black",
        size=12,
    )
```

```
# Show the plot
plt.ylim(0, 1)
plt.show()
```



How Autozen and CBB valuation delta relative to Auction Price changed over time ?

```
# Calculate the delta (difference) between valuation_cbb_trade_in_high and valuation_cbb_t
pred_df["cbb_delta"] = np.abs(pred_df["valuation_cbb_trade_in_high"] - pred_df["valuation_

# Calculate the delta (difference) between valuation_autozen_high and valuation_autozen_lo
pred_df["autozen_delta"] = np.abs(pred_df["valuation_autozen_high"] - pred_df["valuation_a

# Group by quarter year and calculate the average delta for each quarter
quarterly_cbb_delta = pred_df.groupby("quarter_year")["cbb_delta"].mean()
quarterly_autozen_delta = pred_df.groupby("quarter_year")["autozen_delta"].mean()

# Convert Period index to strings
```

```
quarter_labels = quarterly_cbb_delta.index.astype(str)

# Plot the changes in average delta over time
plt.figure(figsize=(10, 6))
plt.plot(quarter_labels, quarterly_autozen_delta.values, color='C1', marker='o', label='Au
plt.plot(quarter_labels, quarterly_cbb_delta.values, color='C0', marker='o', label='CBB De
plt.axhline(pred_df["autozen_delta"].mean(), color='C1', linestyle='--', label='Autozen Av
plt.axhline(pred_df["cbb_delta"].mean(), color='C0', linestyle='--', label='CBB Average')
plt.xlabel("Quarter Year")
plt.ylabel("Average Absolute Delta ($)")
plt.title("Average Absolute Delta ($) over Time")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.show()
```

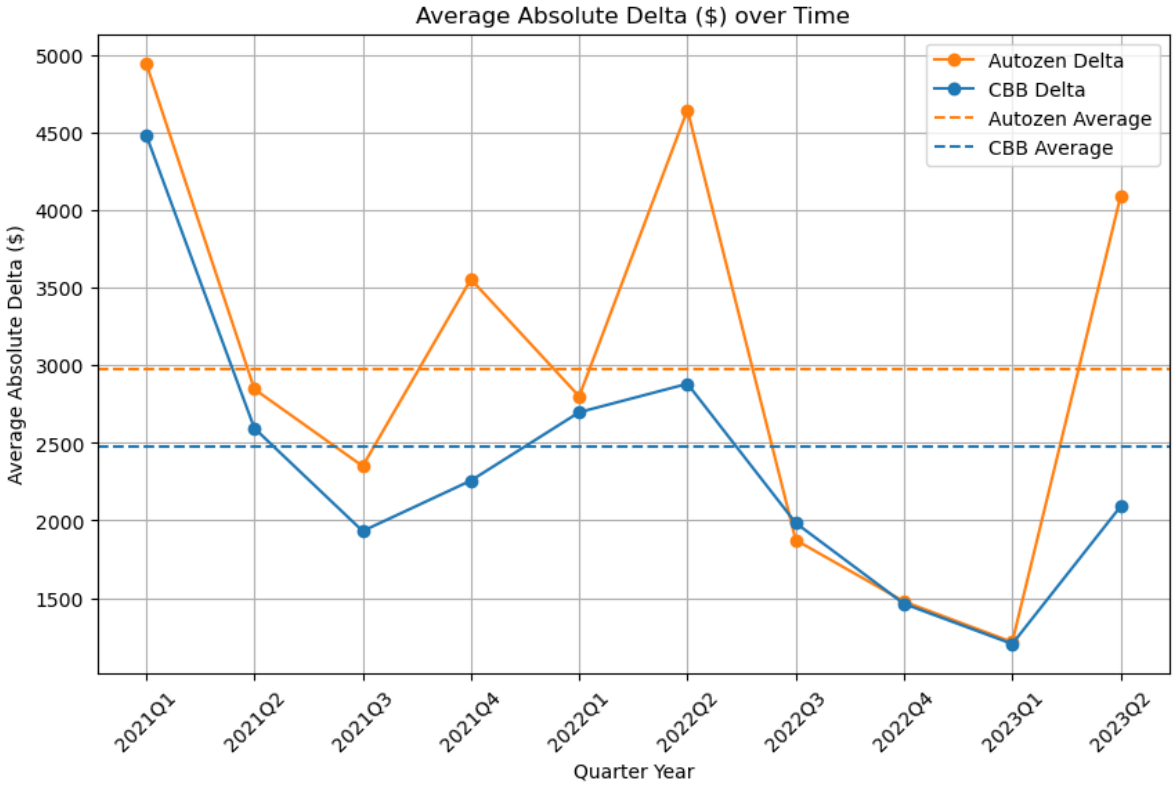


Tableau Dashboard

We also have compiled all these graphs in an interactive Tableau Dashboard for further insights:

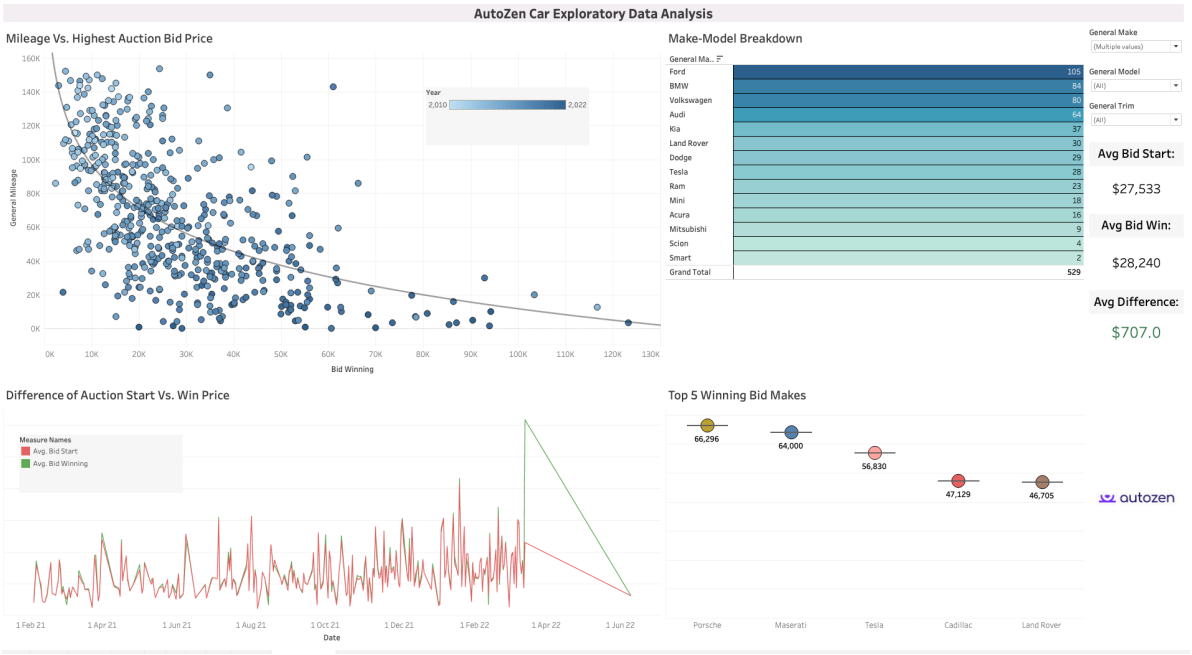


Figure 8: Dashboard