Elian Morales Pina - em486

Collaborator: Thirukumaran Velayudhan - tv95

Professor Michael Houle

DS 675-851

22 October 2023

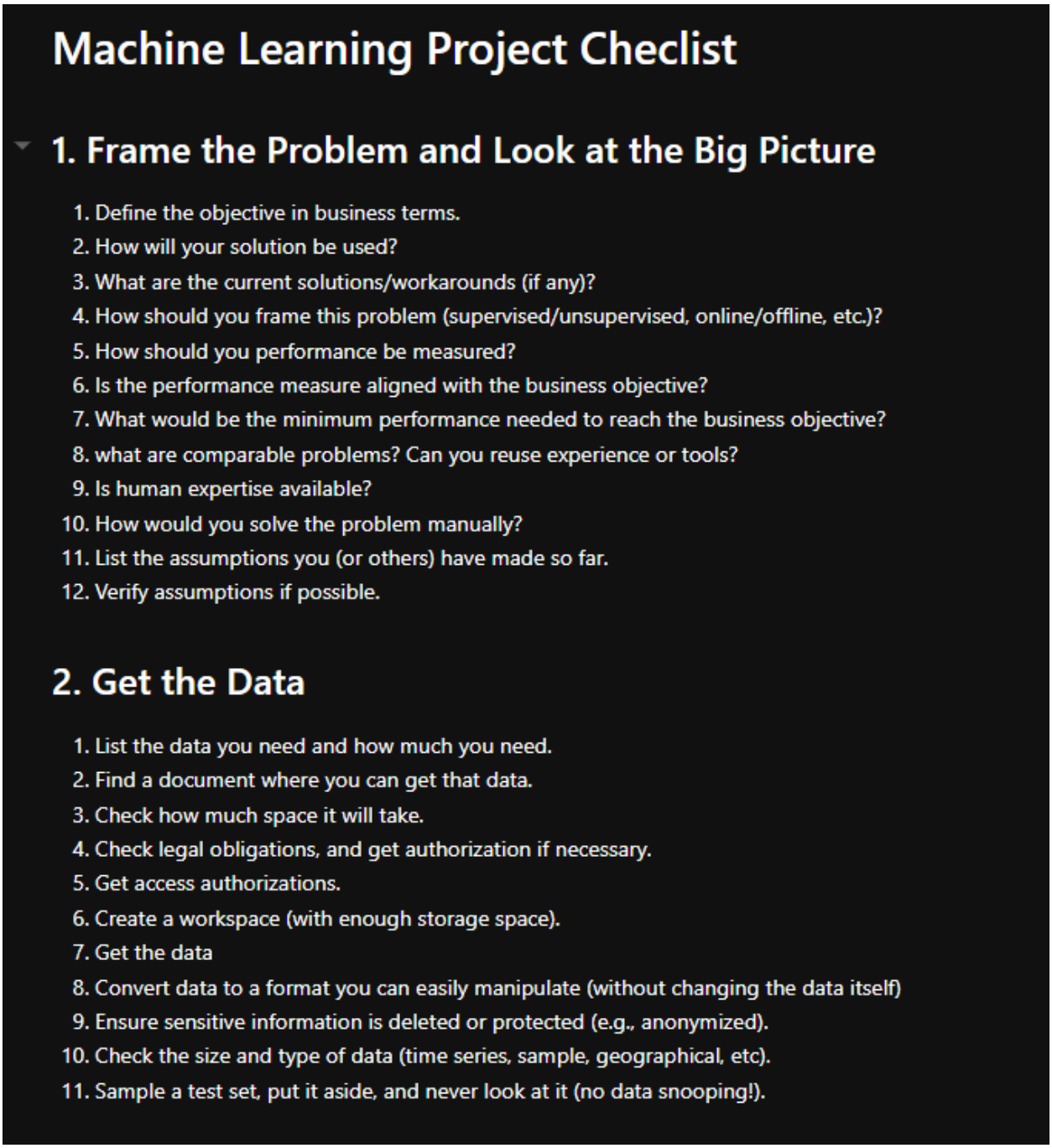
Project Title: DEA - Stellar Classification Dataset - SDSS17

With the availability of vast datasets, data science, and astronomy converge to make sense of this colossal information. I undertook an Exploratory Data Analysis (EDA) journey to achieve stellar classification using the Sloan Digital Sky Survey (SDSS) dataset. The end goal is to determine the best model for accurate prediction. With rigorous EDA and the right prediction model, we can confidently classify stars, galaxies, and quasars, aiding in the broader understanding of our cosmos.

Derived from the SDSS, this dataset boasts 100,000 observations of the cosmos. It offers 17 distinct feature columns that describe each observation and one class column, acting as the definitive label, demarcating observations as a star, galaxy, or quasar. After processing, several algorithms will be tested to see which offers the highest accuracy, precision, recall, and other metrics vital for our classification task. The model with the best metrics will be further fine-tuned for optimization.

Based on the insights from the book "Hands-on Machine Learning with Scikit-Learn & TensorFlow" by Aurélien Géron, my interest in Data Science was piqued. This book laid the foundation for my quest in this field. An essential takeaway from the book, which I fondly refer

to as "My Outline", is the Machine Learning Project Checklist. This structure offers a systematic approach to projects, and I intend to implement it in my project: DEA - Stellar Classification Dataset - SDSS17:



Machine Learning Project Checlist

1. Frame the Problem and Look at the Big Picture

1. Define the objective in business terms.
2. How will your solution be used?
3. What are the current solutions/workarounds (if any)?
4. How should you frame this problem (supervised/unsupervised, online/offline, etc.)?
5. How should you performance be measured?
6. Is the performance measure aligned with the business objective?
7. What would be the minimum performance needed to reach the business objective?
8. what are comparable problems? Can you reuse experience or tools?
9. Is human expertise available?
10. How would you solve the problem manually?
11. List the assumptions you (or others) have made so far.
12. Verify assumptions if possible.

2. Get the Data

1. List the data you need and how much you need.
2. Find a document where you can get that data.
3. Check how much space it will take.
4. Check legal obligations, and get authorization if necessary.
5. Get access authorizations.
6. Create a workspace (with enough storage space).
7. Get the data
8. Convert data to a format you can easily manipulate (without changing the data itself)
9. Ensure sensitive information is deleted or protected (e.g., anonymized).
10. Check the size and type of data (time series, sample, geographical, etc).
11. Sample a test set, put it aside, and never look at it (no data snooping!).

3. Explore the Data

Note: try to get insights from a field expert for these steps.

1. Create a copy of the data for exploration (sampling it down to a manageable size if necessary)
2. Create a jupyter notebook to keep a record of your data exploration.
3. Study each attribute and its characteristics:
    - name
    - Type (categorical, int/float, bounded/unbounded, text, structured, etc.)
    - % of missing values
    - Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)
    - Possible useful for the task?
    - Type of distribution (Gaussian, uniform, logarithmic, etc).
4. For supervised learning tasks, identify the target attribute(s)
5. Visualize the data.
6. Study the correlations between attributes.
7. Study how you would solve the problem manually.
8. Identify the promissing transformations you may want to apply.
9. Identify extra data that would be useful (go back to "Get the Data")
10. Document what you learned.

4. Prepare the Data

Notes:

- Work on copies of the data (keep the original dataset intact).
- write functions for all data transformations you apply, for five reasons:
  - So you can easilly prepare the data the next time you get a fesh dataset
  - So you can apply these transformations in future projects
  - To clean and prepare the test set
  - To clean and prepare new data isntances once your solution is live
  - To make it easy to treat your preparation choices as hyperparameters

1. Data cleaning:
   - Fix or remove outilers (optional)
   - Fill in missing values (e.g., with zero, mean, median...) or drop their rows (or columns).
2. Feature Seelction (optional)
   - Drop the attributes that provide no usefull information for the task.
3. Feature Engineering where appropriate:
   - Discretize continous features.
   - Decompose features (e.g., categorical, date/time, etc.).
   - Add promissing transformations of features (e.g., log(x), sqrt(x), x^2, etc.).
   - Aggregate features into promising new features
4. Feature scaling: standardize or normalize features.

5. Short-List Promissing Models

Notes:

- if the data is huge, you may want to sample smaller training sets so you can train many different models in a reasonable time (be aware that this penalizes complex models such as large neural nets or Random Forests).
- Once again, try to automate these stesps as much as possible.

1. Train many quick and dirty models from different categories (e.g., linear, naive, Bayes, SVM, Random Forests, Neural Net, etc.) using standard parameters.
2. Measure and compare their performance.
    - For each model, use N-fold cross-validation and compute the mean and standard deviation of the peformance measure on the N folds.
3. Analyze the most significant variables for each algorithm.
4. Analyze the type of errors the models make.
    - What data would a human have used to avoid these errors?
5. Have a quick round of feature selection and engineering.
6. Have one or two more quick iterations of the five previous steps.
7. Short-list the top three to five to most promising models, preferrign models that make different types of erros.

6. Fine-Tune the System

Notes:

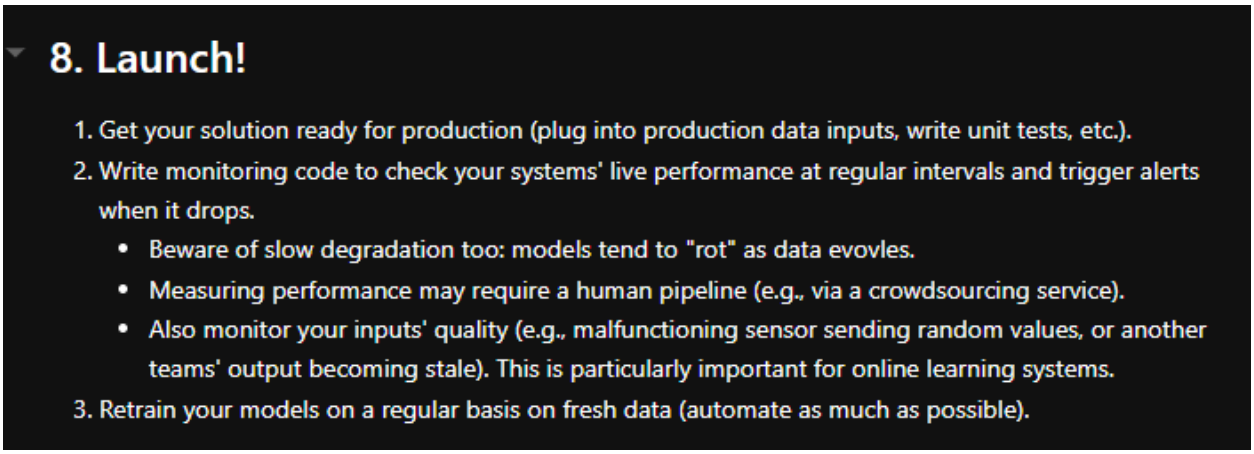
- You will want to use as much data as possible for this step, specially as you move toward the end of fine-tunning.
- As always automate what you can.

1. Fine-tune the hyperparameters using cross-validation.
    - Treat your data transformation choices as hyperparameters, especially when you are not sure about them (e.g., should I replace missing values with zero or with the median value? Or just drop the rows?).
    - Unless there are very few hyperparameter values to explore, prefer random search over grid search. If training is very long, you may prefer a Bayesian optimization approach (e.g., using Gaussian process priors).
2. Try Ensemble methods. Combining your best models will often perform better than running them individually.
3. Once you are confident about your final model, measure its performance on the test set to estimate the generalization error.

Note: Don't tweak your model after measuring the generalization error: you would just start overfittign the test set.

7. present Your Solution

1. Document what you have done.
2. Cerate a nice presentaion.
    - Make sure you highlight the big picture first.
3. Explain why your solution achieves the business objective.
4. Don't forget to present interesting points you noticed along the way.
    - Describe what worked and what did not.
    - List your assumptions and your system's limitations.
5. Ensure your key findings are communicated through beautiful visualizations or easy-to-remenber statements (e.g., "the median income is the number-one predictor of housing prices").



8. Launch!

1. Get your solution ready for production (plug into production data inputs, write unit tests, etc.).
2. Write monitoring code to check your systems' live performance at regular intervals and trigger alerts when it drops.
   - Beware of slow degradation too: models tend to "rot" as data evovles.
   - Measuring performance may require a human pipeline (e.g., via a crowdsourcing service).
   - Also monitor your inputs' quality (e.g., malfunctioning sensor sending random values, or another teams' output becoming stale). This is particularly important for online learning systems.
3. Retrain your models on a regular basis on fresh data (automate as much as possible).

this checklist serves as a framework for a structured approach to machine learning projects, but it's essential to adjust based on the project's specifics and requirements.

In our class, we often didn't delve deeply into data exploration. For the sake of simplicity, we frequently used the familiar iris dataset to clarify intricate topics discussed weekly. As a result, we rarely examined foundational aspects of our data, such as null values or data types (like Numerical or Categorical). I believe these are crucial elements we'll encounter regularly in our professional lives. Hence, I'll be using the guidelines from Geron's textbook to thoroughly explore the data beforehand.

Data Structure

[6]:
```
housing.head()
```

[6]:

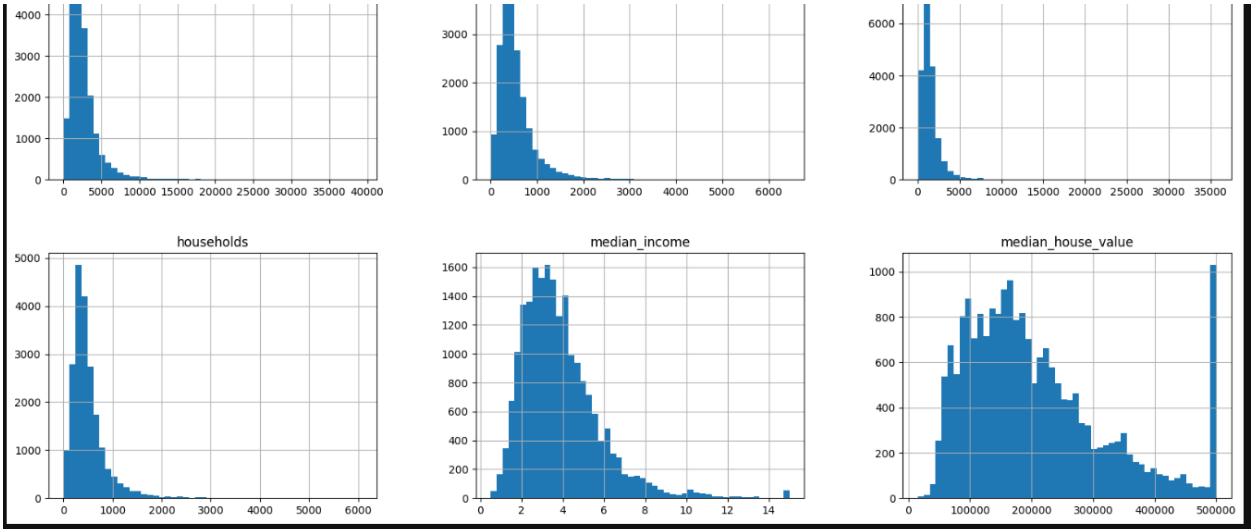
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

[7]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[8]:
```
housing["ocean_proximity"].value_counts()
```

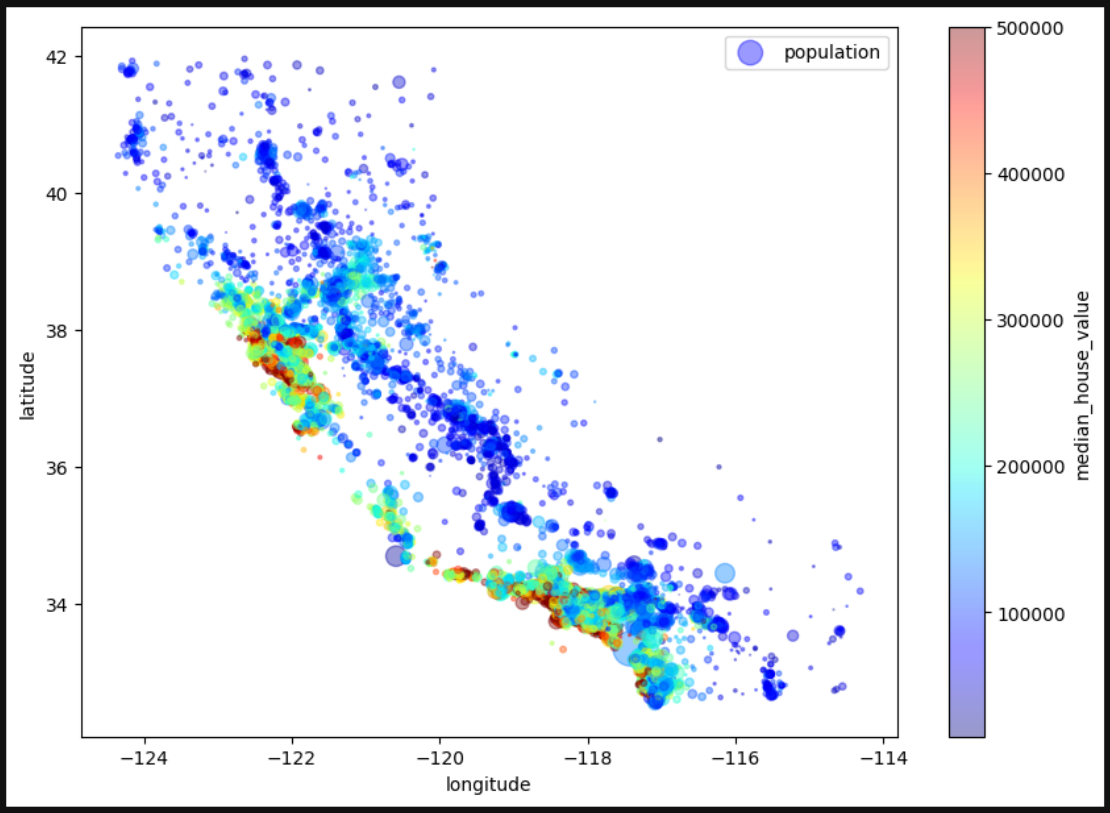
[8]:
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```



1. median income is not expressed in US dollards (15 for higher median incomes and 0.5 for lower median incomes)
2. housing median age has been capped (limited) to 50.0001 years old
3. the median house value has been capped (limited) to 500000.01
    - check with yout team or client to double check if this is prolem or not
    - TODO: if they tell me that they need precise predictions even for above 500,000.01 then:
        - collect proper labels for values above 500,000.01
        - remove those values above 500,000.01
4. Many histograms from above are skewed to right or to the left, this could make harder for machine learning models to detect patterns:
    - therefore, I have to transform these attributes to have a more bell-shaped distributions.

```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population", figsize=(10,7), # Radius of each circle: represent Population
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True, # Color of each circle: represent House Value
    sharex=False)
plt.legend()
```

```
[28]: <matplotlib.legend.Legend at 0x7fb755d1a940>
```



This image tells you that the housing prices are very much related to the location (e.g., close to the ocean) and to the population density.

Recently, in our class, we've begun to adopt pipelines for constructing our machine-learning models. I am particularly a big fan of Geron's approach to using pipelines for data preparation prior to training a machine learning model. Below is a pipeline tailored only for numerical attributes: it fills in missing values using the median, adds new attributes, and scales the data for optimized machine learning model performance:

```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder', CombinedAttributesAdder()), # I could
    ('std_scaler', StandardScaler()),
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

The subsequent pipeline handles both numerical and categorical data. For numerical data, it conducts all necessary preprocessing, and for categorical data, it converts them to one-hot vectors, effectively binarizing each category:

```
num_attribs = list(housing_num) # list of the columns na
cat_attribs = ["ocean_proximity"] # list of the columns

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs), # condcut al
        ("cat", OneHotEncoder(), cat_attribs), # convert
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

To illustrate the efficacy of this comprehensive preprocessing pipeline, I've applied it to a few training data samples:

```
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]

some_data_prepared = full_pipeline.transform(some_data)

print("Predictions", lin_reg.predict(some_data_prepared))
```

```
Predictions [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

It's precisely this seamless integration that I aspire to achieve in my project.

From Walled Faheem's "Stellar Classification and Supervised Learning" Kaggle notebook on stellar data, I observed the potential for insightful data visualizations. These can be instrumental in understanding our dataset. For instance:

Several histograms reveal data skewed either to the right or left. Such skewness can pose challenges for machine learning models when trying to discern patterns. As a result, I'll need to adjust these attributes to achieve distributions that resemble the bell curve more closely.

A lot of machine learning algorithms, notably those dependent on distance measures like k-means or k-nearest neighbors, as well as gradient-driven algorithms like linear regression or neural networks, anticipate features to be zero-centered with uniform variance. For data following a normal distribution, subtracting the mean and dividing by the standard deviation (standardization) can fulfill this expectation.

In typical distributions, approximately 68% of data points fall within one standard deviation from the mean, 95% within two, and 99.7% within three. Data points outside these intervals, especially those beyond 2 or 3 standard deviations, can be viewed as outliers.


0
0
1
2
3
4
5
6
7
redshift

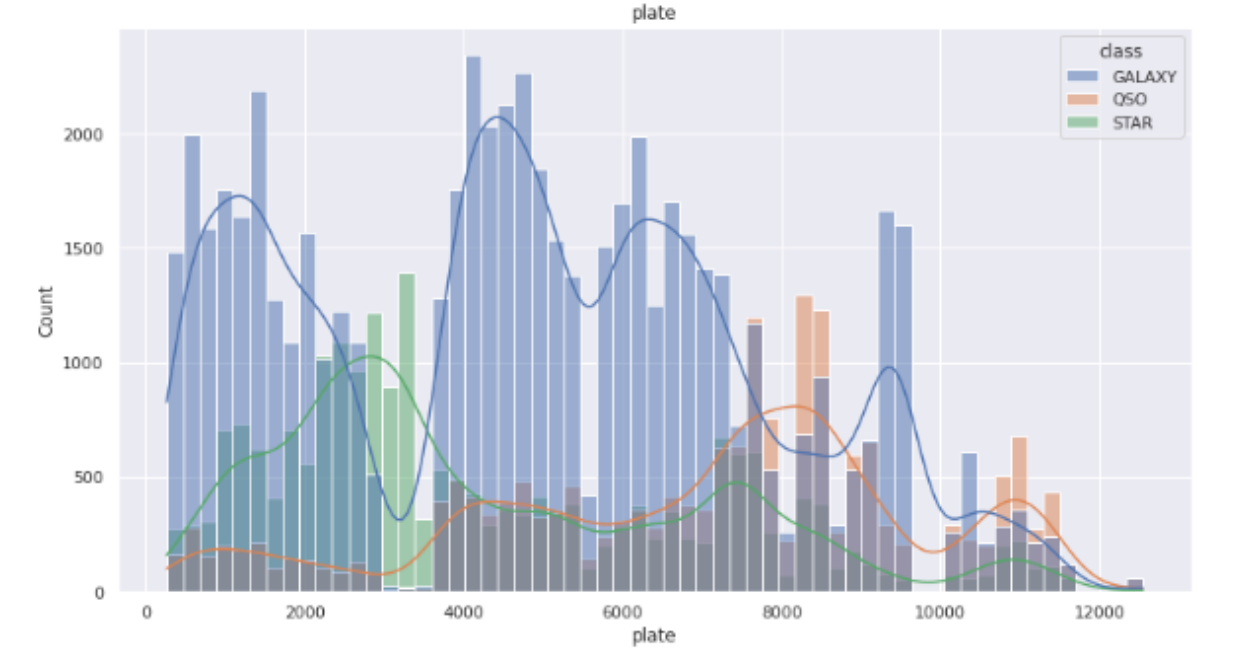
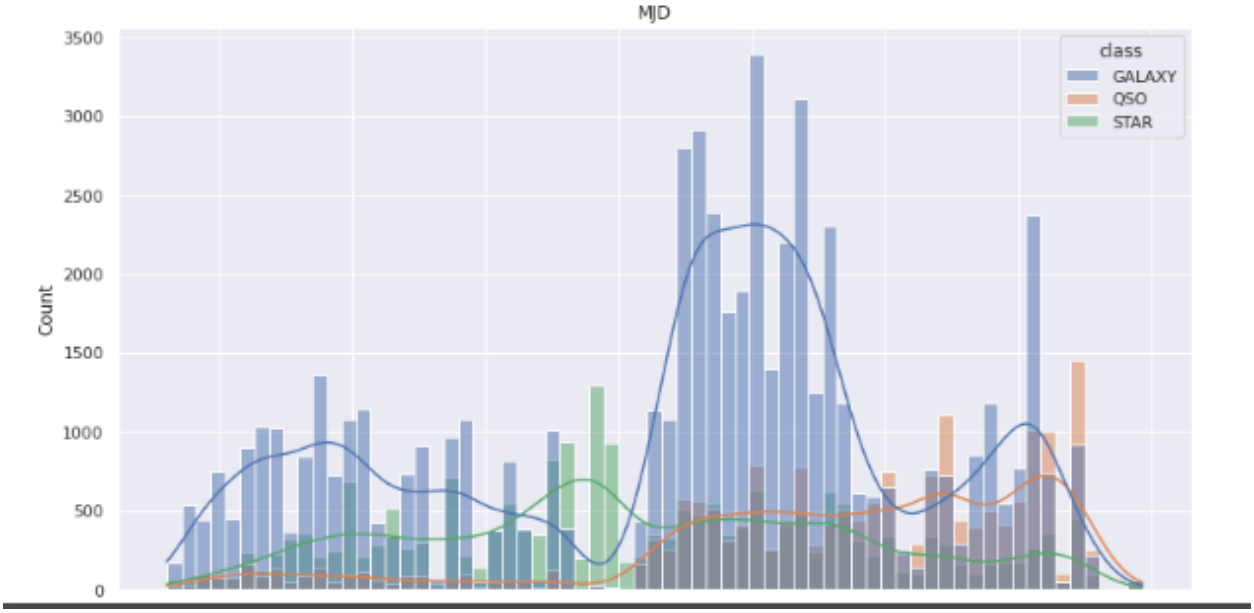


plate
class
GALAXY
QSO
STAR
Count
2000
1500
1000
500
0
0
2000
4000
6000
8000
10000
12000
plate


MJD
class
GALAXY
QSO
STAR
Count
3500
3000
2500
2000
1500
1000
500
0

Dealing with imbalanced data is crucial in various machine learning tasks because the performance of many algorithms can be adversely affected when one class heavily outnumbers the others. When one class is over-represented, many algorithms will be biased towards that class. This means the model might simply predict the majority class most of the time, which can lead to misleadingly high accuracy rates.

Dealing with Imbalanced Data

In [211]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter
```

In [212]:
```
x = df.drop(['class'], axis = 1)
y = df.loc[:,'class'].values
```

In [213]:
```
sm = SMOTE(random_state=42)
print('Original dataset shape %s' % Counter(y))
x, y = sm.fit_resample(x, y)
print('Resampled dataset shape %s' % Counter(y))
```

```
Original dataset shape Counter({0: 50695, 1: 17890, 2: 16158})
Resampled dataset shape Counter({0: 50695, 2: 50695, 1: 50695})
```

The SMOTE function is used to generate synthetic samples in the dataset. The Counter function helps in counting occurrences of unique values, making it useful for displaying class distributions. In the context of imbalanced datasets, "oversampling" refers to the process of increasing the number of instances in the minority class (the class with fewer instances) to

match the number of instances in the majority class (the class with more instances). When you oversample a dataset, you are essentially creating a balanced dataset from an imbalanced one.

Synthetic samples are new data points that are generated programmatically to resemble actual data from the dataset. These aren't duplicates of existing instances but are created based on the features of existing data points. Applies the above SMOTE logic to the datasets x (features) and y (labels). By the end of this process, the dataset will have additional synthetic samples for the minority class, making it balanced with the majority class.

Both Pearson correlation and Spearman correlation are measures of association between two variables. They're used to determine the strength and direction of the linear (Pearson) or monotonic (Spearman) relationship between two variables. In machine learning and data analysis, understanding these correlations can provide insights into the relationships among variables, which can be crucial for feature selection, understanding multicollinearity, and interpreting models.

```
fig = make_subplots(rows=2, cols=1,shared_xaxes=True,subplot_titles=('Perason Correaltion',  'Spearm
an Correaltion'))
colorscale=     [[1.0              , "rgb(165,0,38)"],
                [0.8888888888888888, "rgb(215,48,39)"],
                [0.7777777777777778, "rgb(244,109,67)"],
                [0.6666666666666666, "rgb(253,174,97)"],
                [0.5555555555555556, "rgb(254,224,144)"],
                [0.4444444444444444, "rgb(224,243,248)"],
                [0.3333333333333333, "rgb(171,217,233)"],
                [0.2222222222222222, "rgb(116,173,209)"],
                [0.1111111111111111, "rgb(69,117,180)"],
                [0.0               , "rgb(49,54,149)"]]

s_val =df.corr('pearson')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,name='pearson',showscale=False,xgap=0.7,ygap=0.7),
    row=1, col=1
)


s_val =df.corr('spearman')
s_idx = s_val.index
s_col = s_val.columns
s_val = s_val.values
fig.add_trace(
    go.Heatmap(x=s_col,y=s_idx,z=s_val,xgap=0.7,ygap=0.7),
    row=2, col=1
)
fig.update_layout(
    hoverlabel=dict(
        bgcolor="white",
        font_size=16,
        font_family="Rockwell"
    )
)
fig.update_layout(height=700, width=900, title_text="Numeric Correaltions")
fig.show()
```

The above code from Batucan Senkal “Stars & Galaxies: EDA and Classification” constructs subplots that visually represent Pearson and Spearman correlations of dataset attributes. Using

a heatmap, the intensity and direction of relationships among these attributes are showcased. Recognizing and analyzing these correlations in machine learning can serve various purposes:

- Feature Crafting: It aids in formulating new features that more aptly uncover the inherent trends.
- Model Analysis: Grasping how alterations in one attribute might affect the predicted outcome.
- Dimension Reduction: Pinpointing and either eliminating or merging predictors that have strong correlations.

Works Cited

fedesoriano. (January 2022). Stellar Classification Dataset - SDSS17. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/stellar-classification-dataset-sdss17.

BATUCAN SENKA, Stars & Galaxies: EDA and Classification https://www.kaggle.com/code/psycon/stars-galaxies-eda-and-classification.

BEYZA NUR NAKKAŞ, Stellar Classification - 98.4% Acc 100% AUC https://www.kaggle.com/code/beyzanks/stellar-classification-98-4-acc-100-auc.

WALEED FAHEEM, Stellar Classification and Supervised Learning https://www.kaggle.com/code/waleedfaheem/stellar-classification-and-supervised-learning.

Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 1st ed., O'Reilly Media, Inc., 2017.