# Data Science Notes

**dataset = pd.read_csv("C:/Users/breeh/Downloads/580SurveyCleanup2.csv")** //loading dataset

**subset_data1 = dataset.iloc[:,1:12]** #selecting all rows and columns from 1 to 12

**subset_data1.columns = subset_data1.iloc[0]** //sets the column names of data frame with first row

**subset_data1 = subset_data1[1:]** //this skips the first column and slices thru the index 1

**df_3 = df_2.loc[:, '2.0':'4.0'].replace(1, pd.Series(df_2.columns, df_2.columns))** //replacing the 1s with its column names

**.fillna(" "):** This fills any NaN values in the resulting DataFrame with an empty string

**df_4 = pd.DataFrame(df_3.iloc[:, 0] + df_3.iloc[:, 1], columns=['AIT-580 Section'])** // to concatenate two columns into one and renaming the final concatenated column

**df_5 = pd.concat([df_1, df_4], axis=1)** //concatenating two data frames

**df5.isnull()** //finds all NaN

//Replacing string values with Mode and rest with Median
```
for column in columns_with_missing_values:
   if dataset2[column].dtype == "object":
      dataset2[column].fillna(dataset2[column].mode()[0])
   else:
      dataset2[column].fillna(dataset2[column].median())
```

//To display summary statistics
**df5.describe()**

//To know the DATA TYPES
**df.dtypes**

//Joining two tables
**join_data = pd.merge(data2,data3,on='CustomerID')**

//This counts number of addresses occurences
**data2['Address'].value_counts()**

//Contains tell you if that string is present
**data2['Address'].str.contains('121 Main St')**

//Upper will capitalize it
**data2['State'] = data2['State'].str.upper()**

//Strip removes whitespaces
**data2['Firstname'] = data2['Firstname'].str.strip()**

//To drop duplicates
**data2 = data.drop_duplicates('CustomerID')**

//to drop columns
**column_drop = dataset2.drop(["Access Code", "Email Address"], axis=1)**

Summary of a Column (for Analysis Purposes)

```python
colContent = []
corrected_summary_data = dataset3["AIT-580 Section"]
# corrected_summary_data
temp = corrected_summary_data.describe()
colContent.append(temp)
colContent
```

Python

```
[count    36.000000
 mean      2.888889
 std       1.007905
 min       2.000000
 25%       2.000000
 50%       2.000000
 75%       4.000000
 max       4.000000
 Name: AIT-580 Section, dtype: float64]
```

Since the mean is 2.000000, this means that most students are in section 2, as the median and the 25th percentile indicate the majority concentration in this section

//Column name is Firstname and 2 is row
**data2['Firstname'][2]**

//Creating a new column and assign values to it
**data2['Length'] = data2['Firstname'].map(len)**

```
//Number of missing values in each column
columns_with_missing_values = dataset.isnull().sum()
```

```
### We will use Mode to fill up missing values in Categorical columns
categorical_columns = ['team_position', 'nation_position', 'contract_valid_until', 'joined']
for column in categorical_columns:
   mode_value = column_drop[column].mode()[0]
   column_drop[column] = column_drop[column].fillna(mode_value)
```

```
### We will use mean to fill up missing values in Numerical columns
numerical_mean_columns = ['dribbling', 'defending', 'physic', 'pace', 'shooting', 'passing']
for column in numerical_mean_columns:
   mean_value = column_drop[column].mean()
   column_drop[column] = column_drop[column].fillna(mean_value)
```

```
### We will use median to fill up missing values in Ordinal Numerical columns
numerical_median_columns = ['team_jersey_number', 'nation_jersey_number']
for column in numerical_median_columns:
   median_value = column_drop[column].median()
   column_drop[column] = column_drop[column].fillna(median_value)
```

```
//Dropping values where the value is not equal to GK in a column
column_drop = column_drop.query("nation_position != 'GK'")
or
df= df[df['nation_position'] != 'GK']
```

```
#function to replace + or - with value after performing the operation
def process_skill_value(value):
   # value check if it contains a '+' or '-'
   if isinstance(value, str) and ('+' in value or '-' in value):
      if '+' in value:
         base, increment = value.split('+')
         return int(base) + int(increment)
      elif '-' in value:
         base, decrement = value.split('-')
         return int(base) - int(decrement)
   return int(value)
```

```
//Applying the function to the columns
df['skill_ball_control'] = df['skill_ball_control'].apply(process_skill_value).astype(int)
```

# EDA

# 1. Count of Players by Club Summary
**player_count_by_club = dataset2.groupby('club').size().reset_index(name='player_count').sort_values(by='player_count', ascending=False)**

# 2. Median Team Jersey Number by Club Summary
**median_jersey_number_by_club = dataset2.groupby('club')['team_jersey_number'].median().reset_index().rename(columns={'team_jersey_number': 'median_jersey_number'})**

# 3. Mode of Preferred Foot by Nationality Summary
**mode_preferred_foot_by_nationality = dataset2.groupby('nationality')['preferred_foot'].agg(lambda x: x.mode()[0]).reset_index()**

## Generating Plots

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

### Summary

| Analysis Type | Variables Analyzed | Purpose | Example Plot |
|---|---|---|---|
| **Univariate** | 1 variable | Summarize the distribution or count | Histogram, Box Plot |
| **Bivariate** | 2 variables | Examine relationships | Scatter Plot, Heatmap |
| **Multivariate** | 3+ variables | Explore complex interactions | Pair Plot, 3D Scatter |

**Univariate Analysis**

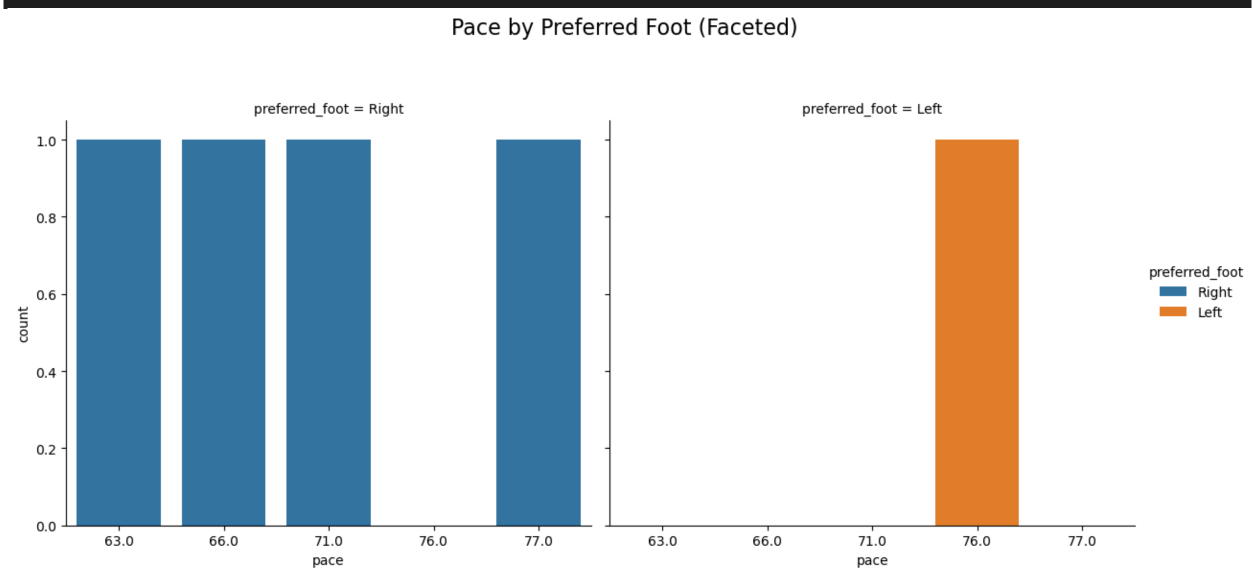
```
# Bar Chart 1: Top 10 distribution of Players by Position
position_counts = dataset2['player_positions'].value_counts().nlargest(10)
plt.figure(figsize=(12, 6))
```

```
sns.barplot(x=position_counts.index, y=position_counts.values,
color='#AEC6CF', edgecolor='black')
plt.title('Top 10 Distribution of Players by Position')
plt.xlabel('Position')
plt.ylabel('Number of Players')
plt.xticks(rotation=90)
plt.show()
```

```
# Histogram 1: Age Distribution of Players
plt.figure(figsize=(10, 6))
sns.histplot(dataset2['age'], bins=10, color='#FFB7CE', edgecolor='black')
plt.title('Age Distribution of Players')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

### Multivariate Analysis

```
subset_df = dataset2[['pace', 'shooting',
'preferred_foot']].dropna().sample(5)
sns.catplot(x='pace', hue='preferred_foot', col='preferred_foot',
data=subset_df, kind='count', height=6, aspect=1)
plt.subplots_adjust(top=0.8)
plt.suptitle('Pace by Preferred Foot (Faceted)', fontsize=16)
plt.show()
```



**Bivariate Analysis**

```
# Scatter Plot 1: Pace vs. Shooting
subset_df = dataset2[['pace', 'shooting']].dropna().sample(100)
plt.figure(figsize=(8, 6))
plt.scatter(subset_df['pace'], subset_df['shooting'])
plt.title('Pace vs Shooting')
plt.xlabel('Pace')
plt.ylabel('Shooting')
plt.show()
```

```
# Box Plot 1: Age by Preferred Foot
plt.figure(figsize=(8, 6))
sns.boxplot(x='preferred_foot', y='age', data=dataset2)
plt.title('Age Distribution by Preferred Foot')
plt.xlabel('Preferred Foot')
plt.ylabel('Age')
plt.xticks(rotation=0)
plt.show()
```

```
# Correlation Plot: Between Height, Weight, and Physic
pastel_cmap = sns.light_palette("skyblue", as_cmap=True)
subset_df = dataset2[['height_cm', 'weight_kg', 'physic']].dropna()
corr_matrix = subset_df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap=pastel_cmap, fmt=".2f")
plt.title('Correlation between Height, Weight, and Physic')
plt.show()
```

**GOT STUCK IN LAST SECTION**

# STATISTICAL INFERENCE

**P < 0.05 we reject the null hypothesis**
**P > 0.05 accept the null hypothesis**

**Q) If math_score of males is greater than females**

H0 = math_score of Males is less than or equal to math_score of Females

H1 = math_score of Males is greater than math_score of Females

```python
males = dataset[dataset['gender'] == 'male']['math_score']
females = dataset[dataset['gender'] == 'female']['math_score']
stats.ttest_ind(males, females, alternative='greater')
```

```
TtestResult(statistic=2.2371559026936256, pvalue=0.012693020299630226, df=1998.0)
```

**Q) If math_score of no part_time_job male students are greater than part_time_job male students.**

Q4

H0 = math_score of no part_time_job male students are less than or equal to math_score of part_time_job male studentsts

H1 = math_score of no part_time_job male students are greater than math_score of part_time_job male students

```python
no_part_time = dataset[(dataset['gender'] == 'male') & (dataset['part_time_job'] == False)]['math_score']
with_part_time = dataset[(dataset['gender'] == 'male') & (dataset['part_time_job'] == True)]['math_score']
stats.ttest_ind(no_part_time, with_part_time, alternative='greater')
```

```
TtestResult(statistic=5.8026672393780085, pvalue=4.3820239750124745e-09, df=996.0)
```

**Q) Is there any Association between gender and career_aspiration**

H0 = no association between gender and career_aspiration

H1 = association between gender and career_aspiration

```
chisqt = pd.crosstab(dataset['gender'], dataset['career_aspiration'],margins=True)
stats.chi2_contingency(chisqt)
chi2, p, dof, ex = stats.chi2_contingency(chisqt)
p
```
[202]

```
1.9987221746002932e-17
```

# REGRESSION

Relationship between a dependent variable (target) and one or more independent variables (predictors). It predicts the value of the dependent variable based on the input variables by identifying patterns and trends in the data.

What does the R-squared value tells us about?

- **Value Range**: $R^2$ ranges from 0 to 1.
  - $R^2 = 0$: The model explains **none** of the variance in the dependent variable (poor fit).
  - $R^2 = 1$: The model explains **100%** of the variance in the dependent variable (perfect fit).
  - $0 < R^2 < 1$: Indicates the percentage of the variance in the dependent variable explained by the model.

1. **High $R^2$ (e.g., > 0.8):**
   - Indicates a strong relationship between predictors and the dependent variable.
   - Commonly seen in physics, engineering, or experimental settings where variables are tightly controlled.
2. **Moderate $R^2$ (e.g., 0.5 to 0.8):**
   - Indicates a moderate relationship.
   - Acceptable in many fields like economics, finance, and social sciences, where data tends to be noisier or influenced by many factors.
3. **Low $R^2$ (e.g., < 0.5):**
   - Indicates a weak relationship.
   - May still be acceptable in fields like psychology or marketing, where behavior is influenced by many unmeasured iables.

What does the p-value tell us about?

**What it means:**

- The p-value is nearly zero, indicating that `GrLivArea` is a statistically significant predictor of `SalePrice`.
- A low p-value (typically < 0.05) means we reject the null hypothesis, confirming that `GrLivArea` significantly impacts `SalePrice`.

// Creating Regression Model
**X = sm.add_constant(data['GrLivArea']) PREDICTOR VARIABLE**
**Y = data['SalePrice'] TARGET VARIABLE**
**model1 = sm.OLS(Y, X).fit()**
**print("Task 1 - Model Summary:")**
**print(model1.summary())**

//finding p-values

**model2.pvalues**

//scatter plot for regression model

```
plt.scatter(data['GrLivArea'], data['SalePrice'], color='blue',
label='Data points')
plt.plot(data['GrLivArea'], model1.fittedvalues, 'r', label='Fitted line')
# plot^ GrLivArea because its predictor
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.title('Regression Line b/w GrLivArea and SalePrice')
plt.legend()
plt.show()
```

// Calculate Correlation

```python
corr1 = data['GrLivArea'].corr(data['SalePrice'])
corr2 = data['TotalBsmtSF'].corr(data['LotArea'])
print("Correlation b/w GrLivArea and SalePrice:", corr1)
print("Correlation b/w TotalBsmtSF and LotArea:", corr2)
```

171] Python

```
Correlation b/w GrLivArea and SalePrice: 0.7086244776126522
Correlation b/w TotalBsmtSF and LotArea: 0.2608331345451576
```

**^Explanation/Reasoning:** There is a significant positive association of roughly 0.709 between `GrLivArea` and `SalePrice`. This indicates that a home's sale price typically rises in proportion to its living area, confirming the significance of living area in home pricing models. By comparison, there is a much weaker positive association approximately 0.261 between `TotalBsmtSF` and `LotArea`. This shows that although there is some correlation between the size of the basement and the lot space, it is quite little and suggests that other factors probably have a greater impact on the lot area.

Correlation is a statistical measure that describes the strength and direction of a relationship between two variables. It tells us how changes in one variable are associated with changes in another.

Positive Correlation : When one variable increases, the other variable also increases.
Negative Correlation : When one variable increases, the other decreases.

**Q) Create regression model to predict SalesPrice using all other inputs**

# finding categorical columns since regression cannot happen on categorical columns

Regression cannot happen on Categorical inputs so convert them to integer / One Hot Encoding

```python
categorical_cols =
data2.select_dtypes(include=['object']).columns.tolist()
data2 = pd.get_dummies(data2, columns=categorical_cols, dtype='int')
print("Data types after converting to dummies just to ensure:")
print(data2.dtypes)
# data2.shape[1]
```

```
X = data2.drop('SalePrice', axis=1)
Y = data2['SalePrice']
X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
model.summary()
```

//Report three MSI and LSI

```
p_values = model.pvalues.sort_values()
print("Most Significant Inputs:")
print(p_values.head(3))
print("\nLeast Significant Inputs:")
print(p_values.tail(3))
# p_values
```

**Q) Create one new input of your choice of values and show the prediction of SalePrice using the same model**

//Creating new input

```
# creating new input
new_input_data = {
    'const': 1, <- ADD THIS EXTRA COLUMN TO NEW DATA INPUT EXCLUDE SALE
PRICE COLUMN FROM HERE
    'LotArea': 9500,
    'OverallQual': 7,
    'OverallCond': 5,
    'YearBuilt': 2001,
    'TotalBsmtSF': 1200,
    '1stFlrSF': 1180,
    '2ndFlrSF': 1100,
    'GrLivArea': 2280,
    'BsmtFullBath': 1,
    'BsmtHalfBath': 0,
    'FullBath': 2,
    'HalfBath': 1,
    'BedroomAbvGr': 4,
    'KitchenAbvGr': 1,
    'Fireplaces': 1,
    'GarageCars': 2,
    'PavedDrive_N': 0,
    'PavedDrive_P': 0,
```

```python
    'PavedDrive_Y': 1,
    'SaleCondition_Abnorml': 0,
    'SaleCondition_AdjLand': 0,
    'SaleCondition_Alloca': 0,
    'SaleCondition_Family': 0,
    'SaleCondition_Normal': 1,
    'SaleCondition_Partial': 0
}

new_input = pd.DataFrame([new_input_data])
new_input
```

```python
model.predict(new_input)
```

From Question 4, drop/remove all the columns which are not signification (p-value >0.05) and create a new model to predict SalePrice. Discuss the performance of the model using few inputs as compared to using all inputs in (Question 4). Which model do you prefer and why? a. The idea is to create a simple generalized model with fewer inputs which are important for prediction and getting the similar performance. For this concept, please research and study "Regularization in Regression"

```
data4 = pd.read_csv('HousePricingData.csv')
if 'Id' in data4.columns:
    data4.drop(['Id'], axis=1, inplace=True)

categorical_cols = data4.select_dtypes(include=['object']).columns.tolist()

data4 = pd.get_dummies(data4, columns=categorical_cols, dtype='int')

# Model from Question 4
X_full = data4.drop('SalePrice', axis=1)
Y = data4['SalePrice']
X_full = sm.add_constant(X_full)
model_1 = sm.OLS(Y, X_full).fit()

# Model with few inputs having p-values < 0.05
significant_vars = model_1.pvalues[model_1.pvalues < 0.05].index.tolist()
X_significant = X_full[significant_vars]
model_2 = sm.OLS(Y, X_significant).fit()
# model_2.summary()
```

```
//Creating three new records
# creating new data
new_input_data = {
    'const': [1, 1, 1],
    'LotArea': [9500, 12000, 8500],
    'OverallQual': [7, 8, 6],
    'OverallCond': [5, 7, 4],
    'YearBuilt': [2001, 1995, 2010],
    'TotalBsmtSF': [1200, 1500, 1100],
    '1stFlrSF': [1180, 1400, 1000],
    '2ndFlrSF': [1100, 1200, 800],
```

```
    'GrLivArea': [2280, 2600, 1800],
    'BsmtFullBath': [1, 2, 1],
    'BsmtHalfBath': [0, 0, 1],
    'FullBath': [2, 3, 1],
    'HalfBath': [1, 1, 0],
    'BedroomAbvGr': [4, 5, 3],
    'KitchenAbvGr': [1, 1, 1],
    'Fireplaces': [1, 2, 0],
    'GarageCars': [2, 3, 1],
    'PavedDrive_N': [0, 0, 1],
    'PavedDrive_P': [0, 1, 0],
    'PavedDrive_Y': [1, 0, 0],
    'SaleCondition_Abnorml': [0, 0, 1],
    'SaleCondition_AdjLand': [0, 1, 0],
    'SaleCondition_Alloca': [0, 0, 0],
    'SaleCondition_Family': [0, 0, 0],
    'SaleCondition_Normal': [1, 0, 0],
    'SaleCondition_Partial': [0, 0, 0]
}

new_input = pd.DataFrame(new_input_data)
new_input
```

# CLASSIFICATION

Do one hot encoding here

Q) Create a train and test set. Consider Admit column as class/label column (Y) and use rest of the columns as inputs (X). Use 30% (test_size=0.3) records for test set. Use the same train and test set for all your analysis with different classifiers

```
X = admission_data.drop(columns=['Admit'])
y = admission_data['Admit']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=99)
print(f"Training X set shape: {X_train.shape}, Test X set shape:
{X_test.shape}, Training y set shape: {y_train.shape}, Test y set shape:
{y_test.shape} ")
```

// Decision Tree

```
X = admission_data.drop(columns=['Admit'])
```

```python
y = admission_data['Admit']
dt_classifier = tree.DecisionTreeClassifier(random_state=99)
dt_classifier.fit(X_train, y_train)
train_pred = dt_classifier.predict(X_train)
print("Training Predictions:\n", train_pred)
fig = plt.figure(figsize=(50, 45))
_ = tree.plot_tree(
    dt_classifier,
    feature_names=list(X.columns),
    class_names=y.value_counts().index.astype(str),
    filled=True
)
plt.title("Decision Tree Visualization for Admission Prediction")
plt.show()
```

//Tree Pruning Analysis