League of Legends Salary Model

Jennifer Dong (jd1358), Haseem Arshad (ha470): GitHub Repo Link

Video Link

## 1. Project Definition

### 1.1 Problem Statement

Stakeholder's Problem: For stakeholders in the League of Legends esports ecosystem knowing the factors influencing a player's salary is key. Salary predictions based on various performance metrics, social influence, and historical data will support important decision-making and facilitate negotiation processes of contracts. This will help teams allocate resources efficiently and help players understand their market value and not to undersell themselves. Aspiring Professional's Problem: We aim to provide insights for players pursuing League of Legends professionally. By identifying the factors most correlated with high player salaries, we can highlight key focus areas, such as metrics on gameplay performance, audience engagement, reach, and overall team/individual player success rates. This information can serve as a guide, helping players understand what it takes to become one of the most paid e-sport athletes.

### 1.2 Connection to Course Material

In summary, our project directly relates to lectures in our class since it requires preparing, collecting records, integrating, transforming, and storing data in a relational database of our own. In more detail: we started by collecting data. Afterward, we can filter out the data using specific cleaning methods, standardization, and ensuring overall structured data. Afterward, we can create a database, which will identify the entities and attributes and design the schema. Afterward, SQL was used to manipulate the database to add records, update records for feature scaling, feature engineering, query for patterns, and query for data visualization. During the duration of the project, we have been applying the fundamentals of Numpy, pandas, and data frame manipulation which have been reinforced during our class. Once we had a database, we normalized our data using one hot and ordinal encoding. We will have to use a linear regression model to train and predict using machine learning techniques discussed in class. Lastly, we will also have to utilize complicated analysis methods using MSE and R^2 from the Scikit library.

## 2. Novelty and Importance

### 2.1 Importance of the Project

Describe the specific challenges it addresses in current data management or related areas. Since the COVID-19 quarantine era, the popularity of esports has been on a big incline. It has become a genuine business in the sports industry and affects the livelihoods of many internationally. For instance, the 2023 League of Legends Worlds final had 6.4 million viewers alone. As a result, understanding player salaries is increasingly relevant at an internal scale for esports companies, brand sponsorships, etc. Additionally, salary predictions are important for prospective players who want to take their League of Legends career paths up to a competitive level and make it into their livelihood. All in all, this project will contribute to creating transparent, data-driven metrics that could shape future player valuations.

### 2.2 Excitement and Relevance

We are super passionate about esports teams such as T1 and Fnatic from League of Legends and are casual players of esports games such as League of Legends, Dota, Fortnite, etc! We find This is a great way to leverage our personal hobbies with the data analysis skills we learned. There is a great potential impact since there are many people in communities (such as Rutgers Esports) that are starting in their esports hobbies so we want to make a digestible way for them to map and scope out the industry demands.

Prior related work?

We are aware of not any work related to our League of Legends Salary Model.

### 2.3 Review of Related Work

Existing data practices in esports generally seem to lack databases for player salary predictors and factors that may contribute to it (such as social media growth, player win rate, etc) all in one. This makes it hard to see data in a cause-and-effect and visual manner for someone outside of the industry. As a result, we want our model and data analysis to be helpful for the esports community as a whole to understand what traits and qualities it takes to make a living in this field of work.

## 3. Progress and Contribution

### 3.1 Data Utilization

We decided to generate synthetic data as the League of Legends community is small and there wasn't sufficient data for the ML portion. Since there wasn't enough public data, Our data was generated through four functions which generated four data frames: player_df, teams_df, socmed_df, and cn_socmed_df: Player stats, team stats, social media and Chinese social media respectively. The reason why we split our data generation into different data frames is because having data that we later joined together on a key would make our data more consistent. For example, our teams_df would hold all team information without making the player_df too convoluted. We have two socmed_dfs in order to mimic real life. Social media apps used in the rest of the world like Twitter and Instagram are unavailable in China so in order to properly gauge a player's social influence, we decided to include data from "Weibo" (on top of apps like "Twitter or Instagram" )which is the primary Chinese social media application. While the two social media data frames were primarily numerical, the teams and player data frame consisted of both numerical and categorical data. Attributes such as account rank tiers, tournament rank tiers, and roles were all data we had to list out and randomly assign to each player.

```python
def generate_socmed_data(start_df, num_records=30):
    # create random seed
    np.random.seed(42)
    random.seed(42)

    # generate the synthetic data
    data = {
        'game_tag': start_df['game_tag'],
        'region': start_df['region'],
        'twitter_followers': np.random.randint(1000, 2000000, num_records),
        'instagram_followers': np.random.randint(1000, 2000000, num_records),
        'livestream_channel_followers': np.random.randint(6000, 500000, num_records),
        'avg_livestream_viewership': np.random.randint(10000, 150000, num_records)
    }

    df = pd.DataFrame(data)


    return df

#creating the dataset
socmed_df = generate_socmed_data(start_df, start_df.shape[0])

#creating a csv out of the dataset
socmed_df.to_csv('socmed_table_initial.csv', index=False)
```

*The full social media function*

```python
def generate_synthetic_lol_data_with_playerstats(num_records=30000):
    # create random seed
    np.random.seed(42)
    random.seed(42)

    #define set data (region, teams, roles, rank of account, tier of the team, set winnings to tier)
    regions_teams = {
        'LCK (KR)': ['T1', 'Gen.G', 'DRX', 'KT Rolster', 'Dplus'],
        'LPL (CN)': ['JD Gaming', 'Bilibili Gaming', 'EDG', 'RNG', 'FPX'],
        'LEC (EU)': ['G2 Esports', 'Fnatic', 'Mad Lions KOI', 'Team BDS', 'GIANTX'],
        'LCS (NA)': ['Cloud9', '100 Thieves', 'FlyQuest', 'Team Liquid', 'Dignitas'],
        'sdfkj': ['Never Sleep', 'Golden Dragon', 'Pajamas Party', '', '']
    }
    roles = ['Top', 'Jungle', 'Mid', 'ADC', 'Support']
    ranks = ['Diamond', 'Master', 'Grandmaster', 'Challenger']

    tier_1 = ['T1', 'Gen.G', 'DRX', 'KT Rolster', 'Dplus', 'JD Gaming', 'Bilibili Gaming', 'EDG', 'RNG', 'FPX', 'G2 Esports', 'Fnatic']
    tier_2 = ['Cloud9', '100 Thieves', 'FlyQuest', 'Team Liquid', 'Dignitas', 'Mad Lions KOI', 'Team BDS', 'GIANTX']

    team_winnings_ranges = {
        'tier_1': (300000, 600000),
        'tier_2': (100000, 300000)
    }

    #functions used to generate player name
    def generate_player_name():
        prefixes = ['Cool', 'Dark', 'Sky', 'Storm', 'Fire', 'Ice', 'Thunder', 'Shadow']
        suffixes = ['Knight', 'Slayer', 'Hunter', 'Master', 'King', 'Lord', 'Warrior']
        numbers = [str(random.randint(0, 999)).zfill(3) for _ in range(1000)]
        return random.choice(prefixes) + random.choice(suffixes) + random.choice(numbers)

    #function used to assign winnings depending on the team level
    def assign_winnings(team):
        if team in tier_1:
            return random.randint(*team_winnings_ranges['tier_1'])
        elif team in tier_2:
            return random.randint(*team_winnings_ranges['tier_2'])
        return random.randint(1000, 7000)

    #making salary directly correlate to team and player win_rate (had to do this or else synthetic data has no correlation)
    def assign_salary_with_winrate(team, win_rate):
        if win_rate >= 0.7:
            salary_range = (700000, 1200000)
        elif win_rate >= 0.5:
            salary_range = (400000, 850000)
        else:
            salary_range = (250000, 600000)

        if team in tier_1:
            salary_range = (salary_range[0] + 100000, salary_range[1] + 200000)
        elif team in tier_2:
            salary_range = (salary_range[0] + 50000, salary_range[1] + 150000)
        else:
            salary_range = (salary_range[0] - 50000, salary_range[1] - 100000)
        return random.randint(*salary_range)
```

Dirtying Dataset:

Since the dataset was generated by us we had to dirty it to give some realism as data collected usually isn't perfect. After we combined all the data frames in SQL into a combined table we dirtied it by randomizing places to put NaNs, duplicating rows, and adding numerical outliers to the data.

```python
#randomly fill with NaNs
def add_nans(df, exclude_cols=None, nan_ratio=0.05):
    """Add NaNs to a DataFrame."""
    dirty_df = df.copy()
    exclude_cols = exclude_cols or []
    for col in dirty_df.columns:
        if col not in exclude_cols:
            mask = np.random.rand(len(dirty_df)) < nan_ratio
            dirty_df.loc[mask, col] = np.nan
    return dirty_df

# adding random duplicate rows
def add_duplicates(df, duplication_ratio_range=(0.03, 0.05)):
    """Add duplicates to a DataFrame."""
    dirty_df = df.copy()
    duplication_count = int(len(dirty_df) * random.uniform(*duplication_ratio_range))
    duplicates = dirty_df.sample(duplication_count, replace=True, random_state=seed)
    dirty_df = pd.concat([dirty_df, duplicates], ignore_index=True)
    return dirty_df

#for numerical data, add outliers
def add_outliers(df, outlier_ratio=0.05):
    """Add extreme outliers to numerical columns in a DataFrame."""
    dirty_df = df.copy()
    numeric_cols = dirty_df.select_dtypes(include=[np.number]).columns
    for col in numeric_cols:
        outlier_count = int(len(dirty_df) * outlier_ratio)
        outlier_indices = np.random.choice(dirty_df.index, outlier_count, replace=False)
        # Generate extreme outlier values (10x the standard deviation)
        std_dev = dirty_df[col].std() if dirty_df[col].std() > 0 else 1
        extreme_values = dirty_df[col].mean() + (10 * std_dev * np.random.choice([-1, 1], outlier_count))
        dirty_df.loc[outlier_indices, col] = extreme_values
    return dirty_df
```

Cleaning the Data:

Our main goal when it came to cleaning the dataset was to make sure it was all usable. Getting rid of NaNs was extremely important to us as we were afraid any NaN would mess with the data visualization and the ML portion. The most important attribute within our table is the game_tag which identifies the players in the dataset. Any rows with NaN as a game_tag were dropped as it meant we had no clue who this player was. For missing regions and teams, we created a mapping from existing data within the table such as which team belonged to which region and vice versa. Using this mapping we were able to fill in some of the team and region data. However, players with both region and team missing had to be dropped due to their placement now being ambiguous. Missing a team is an important factor because it means that their team stats are no longer associated with the player leading to more empty values. After sorting out the three important attributes: game_tag, region, and team we started working through categorical data and numerical data. For categorical attributes in order to fill NaN, we often created mappings based on team or region stats that were available.

For numerical data, we first had to identify and get rid of outliers. Oftentimes, outliers were turned into NaN in order to not interfere with other calculations. For personal player stats, NaNs were often replaced with the median of the players on the same team as they are expected to perform on similar levels. We did this with Z-Score which was very useful. For social media stats like followers, if a player is missing an account then their follower numbers will reflect the team's as many fans of players who don't use social media often follow the team account to get regular updates. There were some stragglers in the end and we used standardization to get rid of that. We used Interquartile Range (IQR) to drop anything that was at the extremes of our data and did not contribute properly to our results.

```
#finding outliers and replacing outliers with NaN
clean_combined_df['zscore'] = (clean_combined_df['team_win_rate'] - clean_combined_df['team_win_rate'].mean())/clean_combined_df['team_win_rate'].std()
clean_combined_df.loc[(clean_combined_df['zscore'] < -3) | (clean_combined_df['zscore'] > 3), 'team_win_rate'] = np.nan


#replacing the NaN with median win rate
clean_combined_df['team_win_rate'] = clean_combined_df['team_win_rate'].fillna(clean_combined_df['team_win_rate'].mean())

#drop zscore col
clean_combined_df.drop(columns=['zscore'], inplace=True)
```

Numerical data cleaning example

## 3.2 Models, Techniques, and Algorithms

We used database querying to identify the fields(columns) that may have correlated with salary the most. Afterward, we used data visualization techniques with Matplotlib to visualize patterns and recurring themes in our data. This is important so we can figure out what factors contribute to more salary. Afterward, we continued with feature engineering to create new fields from preexisting data to map our visualization again and compare correlations. There are some diagrams that show no correlation and some that do show a correlation to salary. Techniques such as hot encoding, and ordinal encoding were helpful. For the attributes: 'team_tournament_rank' and 'region' for example, these attributes are both categorical, and one hot encoding helped us turn them into numerical data which we fed into our linear regression model. Ordinal encoding for example was used on the 'highest_rank' attribute in order to emphasize the importance of some of the ranks.

Using our visualization tools also implemented feature scaling once we encoded our fields so that way we could prepare to train our model on things that were more significant rather than smaller differences in salary. For our model, we used linear regression which uses the attribute: team_tournament_rank, region, win_rate, median_win_rate_by_rank, rank_ordinal_scaled, median_salary_by_team and role_payscale to predict expected salary. Linear regression is super useful since we can train 80% and split 20% for testing.

This allows us to teach the model what traits to look out for more effectively. Throughout the linear regression, we made adjustments to feature scaling based on what we found to be more logical using the data visualization correlation charts.

```
# Define the test cases with 'role' added
test_case = [
    ('S-Tier', 'KR', 'Challenger', 0.72, 'DRX', 'ADC'),  # Added team for median_salary_by_team and role
    ('A-Tier', 'NA', 'Master', 0.60, 'Fnatic', 'Top'),
    ('C-Tier', 'EU', 'Diamond', 0.45, 'Never Sleep', 'Jungle'),
    ('B-Tier', 'KR', 'Grandmaster', 0.65, 'T1', 'Support'),
    ('A-Tier', 'EU', 'Diamond', 0.50, 'Golden Dragon', 'Mid'),
]

# Convert test cases to DataFrame
test_df = pd.DataFrame(test_case, columns=['team_tournament_rank', 'region', 'highest_rank', 'win_rate', 'team', 'role'])

# add 'median_win_rate_by_rank' to test_df using the same mapping from training
test_df['median_win_rate_by_rank'] = test_df['team_tournament_rank'].map(median_winrate_by_rank)

# add 'median_salary_by_team' to test_df using the same mapping from training
test_df['median_salary_by_team'] = test_df['team'].map(median_salary_by_team)

# Ordinal encode 'highest_rank' and scale it
test_df['rank_ordinal'] = test_df['highest_rank'].map(rank_order)
test_df['rank_ordinal_scaled'] = test_df['rank_ordinal'] * 0.1  # Apply scaling as in training

# make 'role_pay' feature based on the given roles
role_pay_mapping = {'ADC': 1.1, 'Top': 1.05, 'Jungle': 1.0, 'Support': 0.95, 'Mid': 0.9}
test_df['role_pay'] = test_df['role'].map(role_pay_mapping)

# encode 'team_tournament_rank'
test_rank_ohe = encoder_rank.transform(test_df[['team_tournament_rank']])
test_rank_df = pd.DataFrame(test_rank_ohe, columns=rank_cols)

# encode 'region'
test_region_ohe = encoder_region.transform(test_df[['region']])
test_region_df = pd.DataFrame(test_region_ohe, columns=region_cols)

# merge all features into the final test DataFrame
test_X = pd.concat([
    test_df[['win_rate', 'median_win_rate_by_rank', 'rank_ordinal_scaled', 'median_salary_by_team', 'role_pay']],  # Added role_pay
    test_rank_df,
    test_region_df
], axis=1)

# making sure our  test cases include the same features and order as x_train
test_X = test_X[x_train.columns]

# making predictions and printing
predicted_salaries = model.predict(test_X)

print("Test Case Predictions:")
for i, case in enumerate(test_case):
    print(f"Input: {case}, Predicted Salary: {predicted_salaries[i]}")
```

We used database tables in order to join our different tables as well as query information from. Each of our visualizations are a result of querying data from the main database tables.

```
# finding the median salary per team
query = """
WITH RankedSalaries AS (
    SELECT
        team,
        salary,
        ROW_NUMBER() OVER (PARTITION BY team ORDER BY salary) AS rank_asc,
        COUNT(*) OVER (PARTITION BY team) AS total_salaries
    FROM cleaned_combined_data_iqr
    WHERE salary IS NOT NULL
)
SELECT
    team,
    AVG(salary) AS median_salary
FROM RankedSalaries
WHERE rank_asc IN ((total_salaries + 1) / 2, (total_salaries + 2) / 2)
GROUP BY team
ORDER BY median_salary DESC;
"""
data = cursor.execute(query).fetchall()

teams = [row[0] for row in data]
median_salaries = [row[1] for row in data]

# making a bar chart
plt.figure(figsize=(27, 6))
bars = plt.bar(teams, median_salaries, color='skyblue', alpha=0.8)
```

## 3.3 Experimental Design

Our prediction was that the higher the winrate of an individual player the more salary they are predicted to earn. While team, region, player's account rank, and their roles may slightly affect the predicted outcome we both believed that the majority of weight for a player's salary would be their win rate as win rate is what truly demonstrates the capabilities of a player. Some roles may change the expectations for each player but the overall consensus is that the more stable a player is in terms of being able to win games and tournaments is what makes them valuable to a team. Win Rate is the most numerical way to define a player's achievements in game and is something teams definitely look at when determining their salaries.

## 3.4 Key Findings and Results

The results were quite similar to what we had originally hypothesized, with the player with the highest winrate making the most.

```
Test Case Predictions:
Input: ('S-Tier', 'KR', 'Challenger', 0.72, 'DRX', 'ADC'), Predicted Salary: 880791.6282184008
Input: ('A-Tier', 'NA', 'Master', 0.6, 'Fnatic', 'Top'), Predicted Salary: 755244.5529775032
Input: ('C-Tier', 'EU', 'Diamond', 0.45, 'Never Sleep', 'Jungle'), Predicted Salary: 419370.4610486062
Input: ('B-Tier', 'KR', 'Grandmaster', 0.65, 'T1', 'Support'), Predicted Salary: 801732.9266814538
Input: ('A-Tier', 'EU', 'Diamond', 0.5, 'Golden Dragon', 'Mid'), Predicted Salary: 533633.4963451284
```

## 3.5 Evaluation

We used mean squared error and r-squared to evaluate our predictions. It did not meet our desires but it was expected due to our limitations below. The R^2 value is under 50 which means our model was not fitted with data for ex.

```python
#reprinting metrics
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
from sklearn.metrics import mean_squared_error, r2_score

y_test_pred = model.predict(x_test)
print("MSE:", mean_squared_error(y_test, y_test_pred))
print("R²:", r2_score(y_test, y_test_pred))
```

```
Model Coefficients: [ 9.28373979e+05 -7.25913561e+02  3.26454080e+03  5.73832530e-01
  1.13837981e+04  1.45655268e+04  2.04010451e+04 -5.89644045e+04
  2.39978325e+04  9.91571378e+04 -8.15369341e+04 -1.76202037e+04]
Model Intercept: -220213.88792108558
MSE: 24021235575.92781
R²: 0.3370303350743864
```

### 3.6 Advantages and Limitations

The biggest limitations with this approach is that our dataset is randomized meaning several player's stats and numbers are not correlated. We did try to influence a little but of the script generation to keep it within reason but largely because of randomization there was little to no correlation in our data for our model to be trained on. Potential improvements would be to get a lot of data from a lot of different players over a longer period of time in order to see salary improvements and salary's correlation to other variables through our original plan of web scraping. This approach lets us see how well linear regression works when it comes to predictions as our numbers are currently standard compared to the more fluid numbers of real life. Our current method puts more emphasis on player's in-game stats and it would be interesting to see if with real data the linear regression will rely on another category.

## 4. Changes After Proposal

### 4.1 Differences from Proposal

In our proposal, our main focus for data collection was web scraping. However, after a few days of scraping and trying to use APIs, we realized that quite frankly we did not have enough records after data scraping and the records we did scrape took so long to acquire and from many public websites as well. Although we were able to get API access for some social media sites the majority of the actual information about player game stats, tournament stats, and the actual esport was trapped behind web scraping only since APIs were not available.

We had originally scrapped only 300 rows using websites such as Esport Earnings, Liquipedia, and Esport Charts, before realizing that League of Legends esports was an incredibly niche topic to find data about. Additionally due to site restrictions and rate limiting we were led into a roadblock where we either had to choose between a small dataset or change the data collection plan. We decided to pivot and generate synthetic data in order to stay on course with our original salary model with new data. This was the primary difference between the original and proposed idea since we did not web scrape. However, using synthetic data proposed new challenges as we had to now ensure realism despite randomly generating our data.

### 4.2 Bottlenecks and Challenges

As mentioned earlier, We spent a lot of time trying to figure out whether web scraping certain websites was even allowed and most of the time it wasn't. There were limitations to official APIs where we didn't have the access levels to get the data we wanted. We tried many ways to web scrape but the process was

too slow especially when we were trying to pull 500+ players' information without getting rate limited. We had tried timers but the thing is it would take way too long to send 1000 query requests in 10+ second intervals just for most of the information to not be there. Eventually, my partner and I realized that the data we had wasn't substantial enough so we eventually decided to generate synthetic data. However, we did spend several days web scraping and we both learned a lot about web scraping a website. We found ways to avoid being rate-limited as well as how to navigate through a website's HTML. Additionally, from synthetic data generation, we had to backtrack after already starting because we realized that since our data was randomly generated in numpy we had no correlation to prove. As a result, we researched the most typical factors and drew small correlations in the data and trained our model on those correlations.

However, due to the randomness of the data, this was difficult to make sure we were not outright rigging the whole data. If we did rig the whole data then it would not be realistic in real life. As a result, the fact that our R^2 is not too close to 1 is good since we are not overfitting (or rigging our model) to only fit our singular randomized data . We chose to train our model based on patterns observed but primarily applicable in real life as well so that way it can be generalized regardless of the MSE or R^2 value stated. Overall due to the scale of the community on which we did our data reports, there were many challenges, yet we learned to be resilient, communicate well, and do our best to research.

## 5. Conclusion and Future Work

### 5.1 Summary of Contributions

From start to finish we communicated well as a duo and did our best to bounce off each other's work. We would call virtually to ensure we split up the work. To start we split up the pre, and post report. We split up the initial web scraping (although we scrapped this it still took a few days out of our work time). Afterward, when we switched to using synthetic data we were rotating which functions to write. For example: Jennifer would write the data generation and Haseem would follow up with the data dirtying and we would keep alternating. When one of us was stuck we would work on the same portion together to ensure seamless progress and communication during the project.

### 5.2 Future Directions

In the future what we can do is extend our research using real data as mentioned earlier. This would be a long process since it would require web scraping from many websites and potentially requesting/purchasing APIs so we can get data without rate limits. Additionally we can move to other esport games such as Dota or Counter Strike which are additionally played by millions. Lastly, we can make our own website with a front end interface to allow visualization and salary pattern predictions across many esport games. This would of course take a long time scraping and combining data for each sport and making the website as well.