Salary Prediction

This project aims to predict average salary for a given job using machine learning. Users can enter job and company details via a Streamlit web application, and the model predicts salary accordingly.

Among the models tested, Random Forest performed best and was selected as the final model for accurate salary prediction.

The dataset has been taken from Kaggle: Jobs Dataset from Glassdoor. It contains job postings with following attributes:

Column Name	Description
job_id	Unique identifier for the job posting
job_state	State where the job is located
same_state	Binary indicator if job is in the same state as the user
age	Age of the person looking at the job
python_yn	Binary indicator if the person knows Python
R_yn	Binary indicator if the person knows R
spark	Binary indicator if the person knows Spark
aws	Binary indicator if the person knows AWS
excel	Binary indicator if the person knows Excel
job_simp	Simplified job title
seniority	Seniority level of the job

The target variable is avg_salary, which represents the average salary for the job posting.

Data Loading

In [53]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
#load the dataset and display it
df = pd.read_csv('eda_data.csv')
df
```

Out[ ]:

	Unnamed: 0	Job Title	Salary Estimate	Job Description	Rating	Company Name	Location	Headquarters
0	0	Data Scientist	$53K−91K (Glassdoor est.)	Data Scientist\nLocation: Albuquerque, NM\nEdu...	3.8	Tecolote Research\n3.8	Albuquerque, NM	Goleta, CA
1	1	Healthcare Data Scientist	$63K−112K (Glassdoor est.)	What You Will Do:\n\nI. General Summary\n\nThe...	3.4	University of Maryland Medical System\n3.4	Linthicum, MD	Baltimore, MD
2	2	Data Scientist	$80K−90K (Glassdoor est.)	KnowBe4, Inc. is a high growth information sec...	4.8	KnowBe4\n4.8	Clearwater, FL	Clearwater, FL

Data Preprocessing/Cleaning

1. Checked all column names to understand the dataset structure.

In [5]:
```
#show available columns
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'Job Title', 'Salary Estimate', 'Job Description',
       'Rating', 'Company Name', 'Location', 'Headquarters', 'Size', 'Founded',
       'Type of ownership', 'Industry', 'Sector', 'Revenue', 'Competitors',
       'hourly', 'employer_provided', 'min_salary', 'max_salary', 'avg_salary',
       'company_txt', 'job_state', 'same_state', 'age', 'python_yn', 'R_yn',
       'spark', 'aws', 'excel', 'job_simp', 'seniority', 'desc_len',
       'num_comp'],
      dtype='object')
```

2. Dropped columns that are irrelevant for prediction

In [6]:
```
#drop unnecessary or redundant columns that are not useful for prediction
df.drop(columns=[
    'Unnamed: 0', 'Job Title', 'Salary Estimate', 'Job Description',
    'Location','Headquarters','company_txt', 'Competitors','min_salary',
    'max_salary', 'desc_len', 'same_state','employer_provided','Company Name'
],inplace=True)
```

3. Renamed columns for readability

In [7]:
```
#rename columns for readability
df.rename(columns={
    'job_simp': 'Job Role',
    'num_comp': 'Number of Competitors',
    'seniority': 'Seniority Level',
    'R_yn': 'R Required',
    'python_yn':'Python Required',
    'spark':'Spark Required',
    'aws':'AWS Required',
    'excel':'Excel Required'
}, inplace=True)
```

4. Checked for missing values

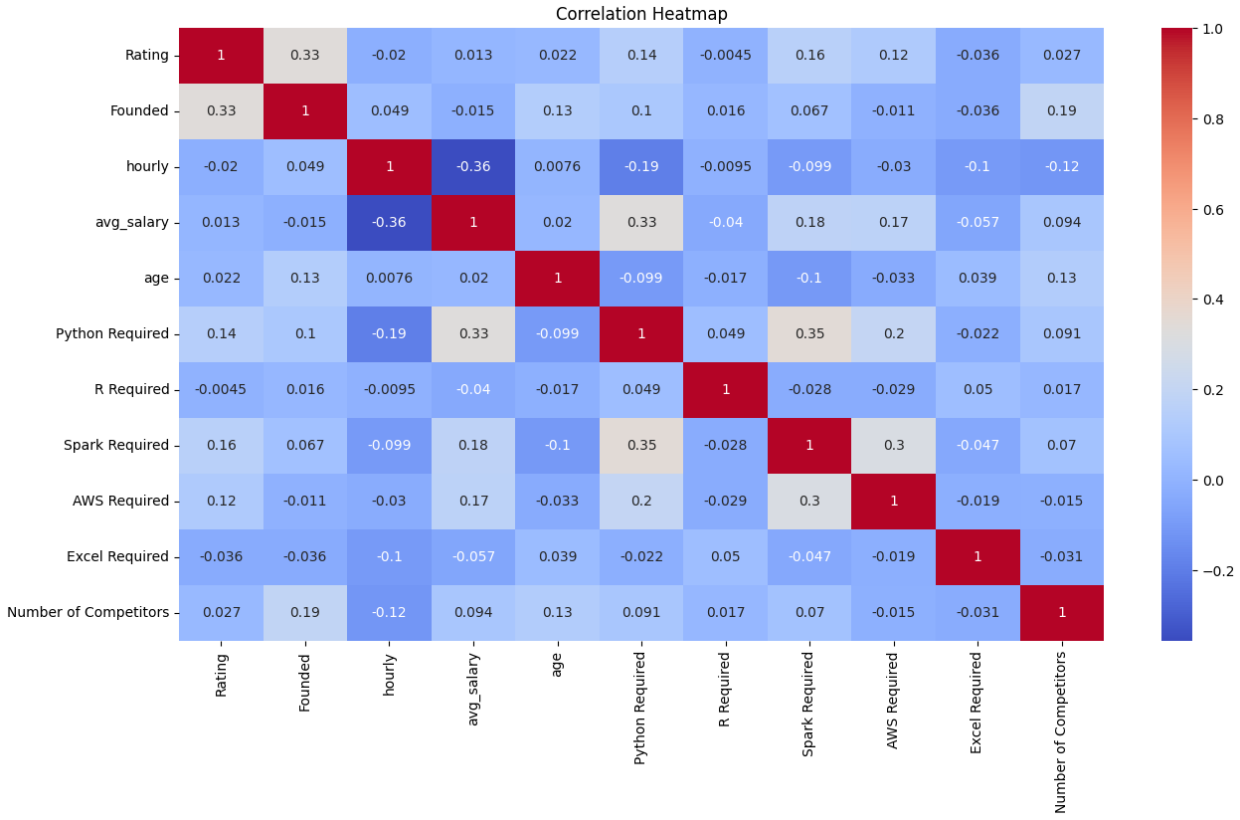
In [9]:
```
#check for missing values
df.isnull().sum()
```

Out[9]:
```
Rating                    0
Size                      0
Founded                   0
Type of ownership         0
Industry                  0
Sector                    0
Revenue                   0
hourly                    0
avg_salary                0
job_state                 0
age                       0
Python Required           0
R Required                0
Spark Required            0
AWS Required              0
Excel Required            0
Job Role                  0
Seniority Level           0
Number of Competitors     0
dtype: int64
```

Exploratory Data Analysis (EDA)

1. Correlation Heatmap

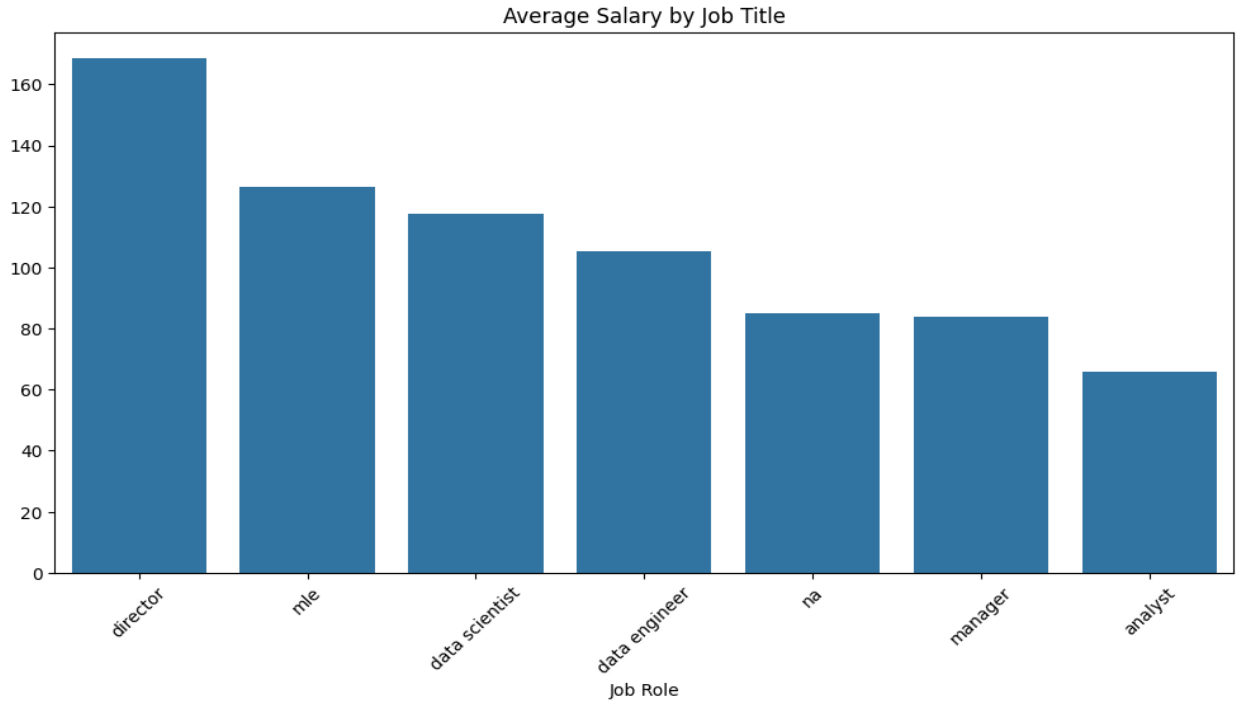
```
In [10]: #correlation heatmap of numerical features
corr = df.select_dtypes(include=[np.number]).corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr,annot=True,cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



From the above fig, we can see multiple numerical features collectively affect salary prediction. Like, slight positive correlation with Python Required (0.33) and Spark Required (0.18) suggest jobs requiring these skills tend to have higher salaries. Thus, it concludes, no single variable dominates, highlighting the importance of using an ensemble model like random forest to capture complex interactions.

2. Average Salary by Role

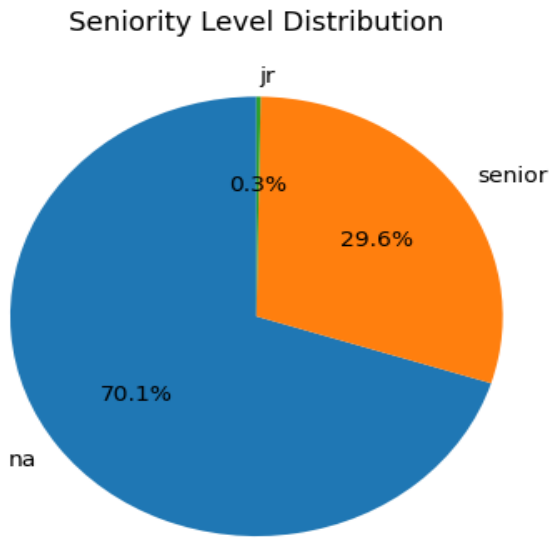
```
In [11]: #average salary by job role
df_sorted = df.groupby('Job Role')['avg_salary'].mean().sort_values
plt.figure(figsize=(12,6))
sns.barplot(x=df_sorted.index, y=df_sorted.values)
plt.title("Average Salary by Job Title")
plt.xticks(rotation=45)
plt.show()
```



This bar chart shows that higher level roles like director have the highest salary, then MLE and data scientist earn moderately high salaries, after data engineer earns slightly less, and manager and NA are paid lower while entry or mid-level roles like analyst pay less.

3. Distribution of Seniority Level

```
In [12]: #distribution of seniority levels
seniority_counts = df['Seniority Level'].value_counts()
plt.pie(seniority_counts, labels=seniority_counts.index, autopct='%1.1f%%'
plt.title("Seniority Level Distribution")
plt.show()
```



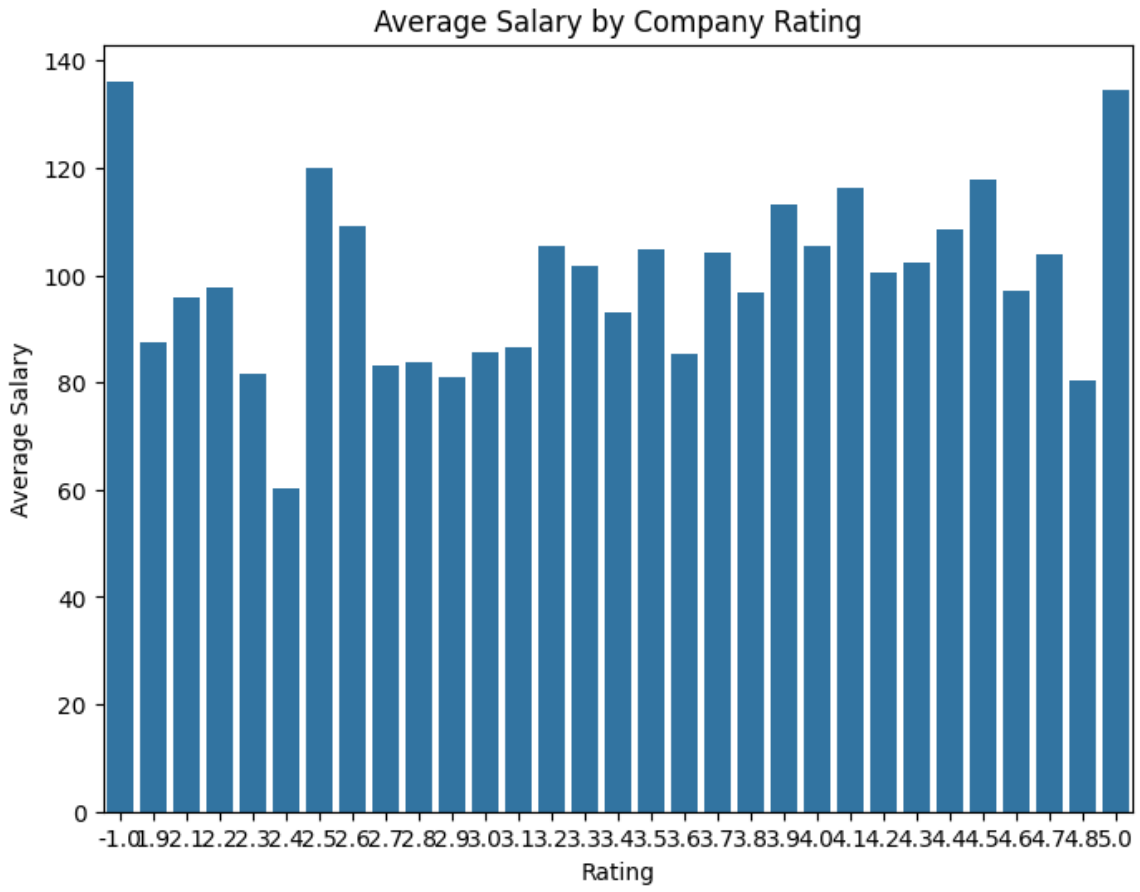
This pie chart shows the distribution of seniority levels:

- 70.1% have no seniority level recorded(na)
- 29.6% are senior and only 0.3% are junior.

So most people don't have a seniority level marked, and among those who do, seniors are far more common that juniors.

4. Average Salary by Company Rating

```
In [56]: avg_salary = df.groupby('Rating')['avg_salary'].mean().
         plt.figure(figsize=(8,6))
         sns.barplot(x='Rating', y='avg_salary', data=avg_salary
         plt.title("Average Salary by Company Rating")
         plt.xlabel("Rating")
         plt.ylabel("Average Salary")
         plt.show()
```



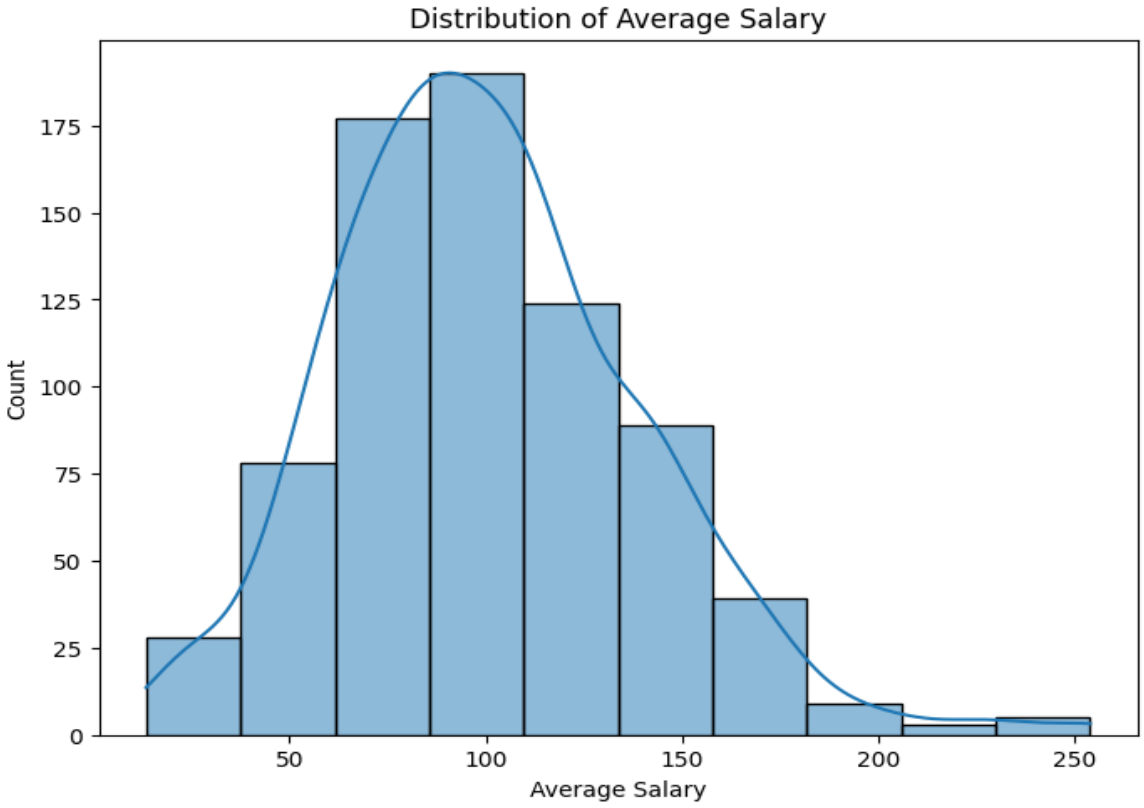
From this chart we can see,

- Salaries don't steadily increase with higher ratings – instead, they go up and down, with some ratings having unexpectedly higher or lower salaries.

So, salary is not strongly linked to company rating here, it varies across ratings.

5. Distribution of Average Salary

```
In [57]: #distribution of average salary
         plt.figure(figsize=(8,6))
         sns.histplot(df['avg_salary'],bins=10, kde=True)
         plt.title("Distribution of Average Salary")
         plt.xlabel("Average Salary")
         plt.ylabel("Count")
         plt.show()
```



This histogram shows ,

- The peak salaries are around 90-100 meaning most employees earn in that range.
- Fewer people earn very low (<40) or very high (>160) salaries.

The KDE line shows its roughly bell-shaped but slightly skewed to the right (a few very high salaries pull the tail). So most salaries are moderate (around 90-100), with fewer very high salaries.

Encoding Categorical Features

In [14]:
```
#encode categorical seniority level into numeric form
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Seniority Level'] = le.fit_transform(df['Seniority Level'])
```

In [15]:
```
#one-hot encoding of categorical features; convert into dummy var
df = pd.get_dummies(df, columns=['Job Role', 'Size', 'job_state',
```

- Here, Label encoding is used for seniority level to preserve the order (Junior<Mid<Senior).
- While, One hot encoding is applied to other categorical variables (job role, company size, etc.) to convert them into dummy variables as they have no natural order.

Defining Features and Splitting Dataset

In [16]:
```
#define features(X) and target(y)
X = df.drop('avg_salary',axis=1)
y = df['avg_salary']
```

In [17]:
```
#split dataset into training and testing set(80% train, 20% test)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

Feature Scaling

In [35]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Standardized the features using StandardScaler so that each feature has a mean of 0 and standard deviation of 1.

Model training, evaluation, comparison

1. Random Forest Regressor

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n_estimators=100, random_state=42)
#train model
rfr.fit(X_train, y_train)
```

In [37]:
```
#predict on test data
y_pred = rfr.predict(X_test)
```

2. Model Evaluation

In [38]:
```
#calculate evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error: {rmse}")
r2 = r2_score(y_test, y_pred)
print(f"R^2 Score: {r2}")
```

```
Mean Squared Error: 358.03961023431634
Root Mean Squared Error: 18.92193463243958
R^2 Score: 0.7809500407668317
```

3. Comparison with other models

In [39]:
```
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

models = {
    "Random Forest": rfr,
    "XGBoost": XGBRegressor(n_estimators=100, random_state=42),
    "Linear Regression": LinearRegression(),
    "K-Nearest Neighbors": KNeighborsRegressor(n_neighbors=5),
    "Support Vector Regression": SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1),
}

results={}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    results[name] = {"MSE": mse, "RMSE": rmse, "R^2": r2}

results_df = pd.DataFrame(results).T
results_df = results_df.round(4)

print(results_df)
```

	MSE	RMSE	R^2
Random Forest	358.0396	18.9219	0.7810
XGBoost	447.0567	21.1437	0.7265
Linear Regression	766.6551	27.6885	0.5310
K-Nearest Neighbors	1076.3790	32.8082	0.3415
Support Vector Regression	569.0690	23.8552	0.6518

- Trained and evaluated multiple regression models(XGBoost, linear regression, KNN, SVR) and compared their performance using MSE, RMSE and R2.
- The o/p shows random forest is our best model for predicting salary as it has highest R^2 score and lowest error.

Cross-Validation

In [58]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(rfr, X, y, cv=5, scoring='r2')
print(f"Cross-Validation R^2 Scores: {cv_scores}")
print(f"Mean Cross-Validation R^2 Score: {cv_scores.mean()}")
```

```
Cross-Validation R^2 Scores: [0.47688931 0.64007616 0.82180422 0.81984521 0.61155014]
Mean Cross-Validation R^2 Score: 0.6740330092271417
```

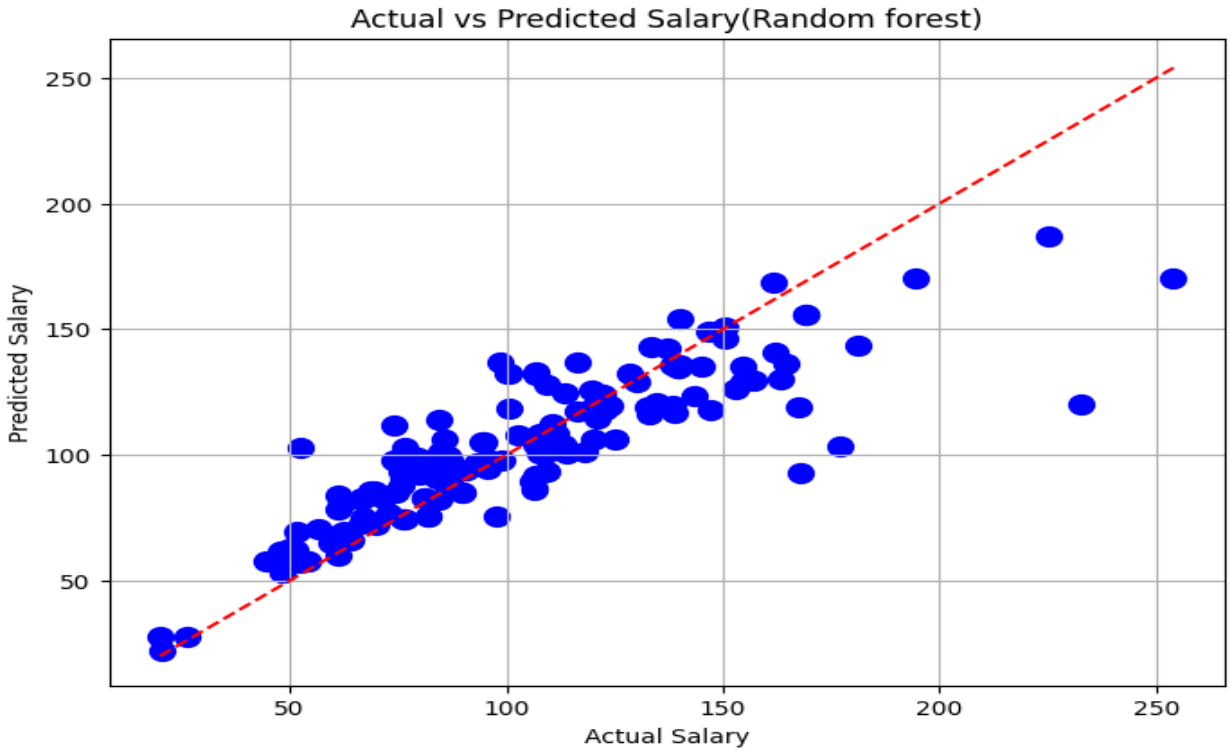
- Here, we can see mean r square is 0.67 so the random forest model performs moderately well capturing most patterns in salary.

Hyper parameter Tuning

In [59]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distributions = {
    'n_estimators' : randint(100,300),
    'max_depth' : [None,5,10,15],
    'min_samples_split' : [2,5,10],
    'min_samples_leaf' : [1,2,4],
    'max_features':['sqrt','log2',None]
}

random_search = RandomizedSearchCV(
    rfr,
    param_distributions=param_distributions,
    scoring='r2',
    cv=3,
    n_jobs=-1,
    random_state=42
)

random_search.fit(X_train, y_train)
best_model = random_search.best_estimator_
print("Best parameters:", random_search.best_params_)
```

Visualization of prediction

```
In [60]: #Visualization of predictions
y_pred_final = best_model.predict(X_test)

plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred_final, color='blue',s=100)
plt.plot([min(y_test),max(y_test)],[min(y_test),max(y_test)],'r--')
plt.xlabel('Actual Salary')
plt.ylabel('Predicted Salary')
plt.title('Actual vs Predicted Salary(Random forest)')
plt.grid(True)
```



Here,

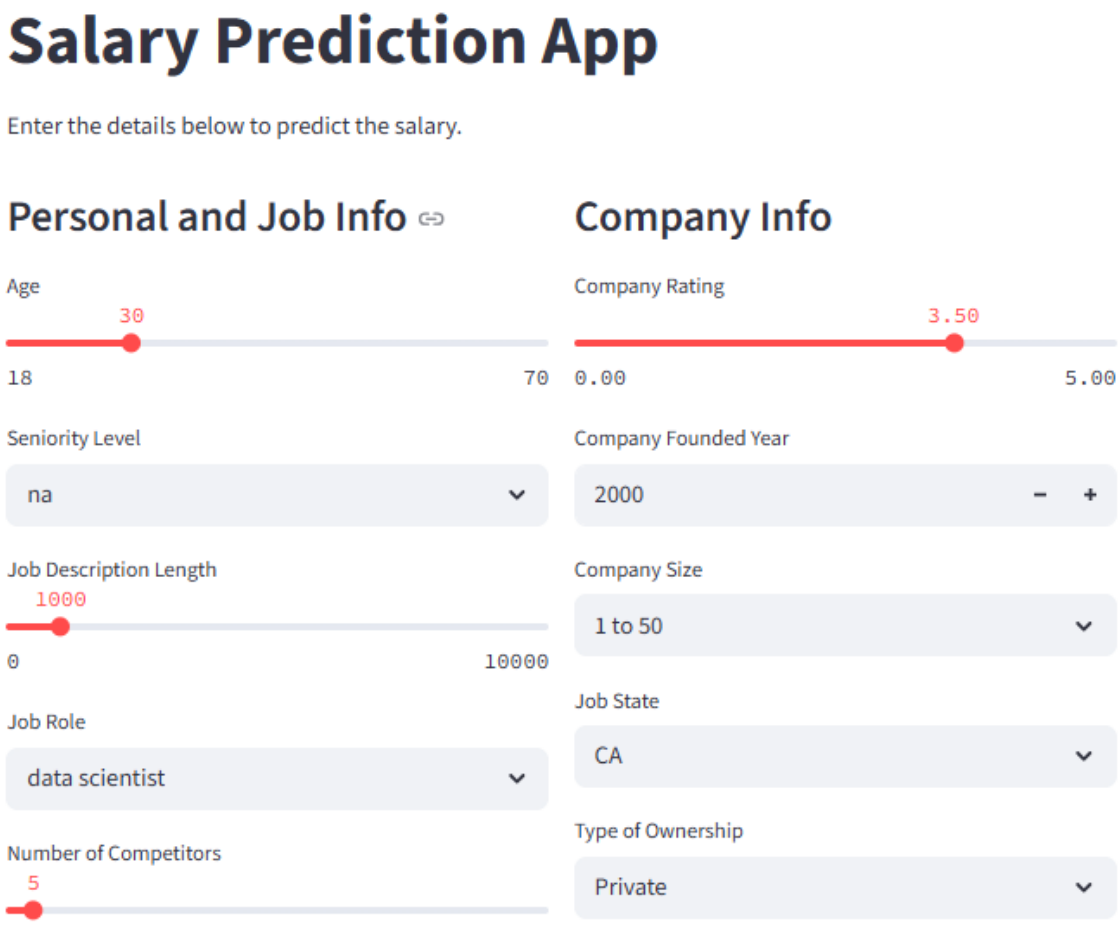
- Red dashed line represents perfect prediction.
- Points close to this line indicate accurate predictions.

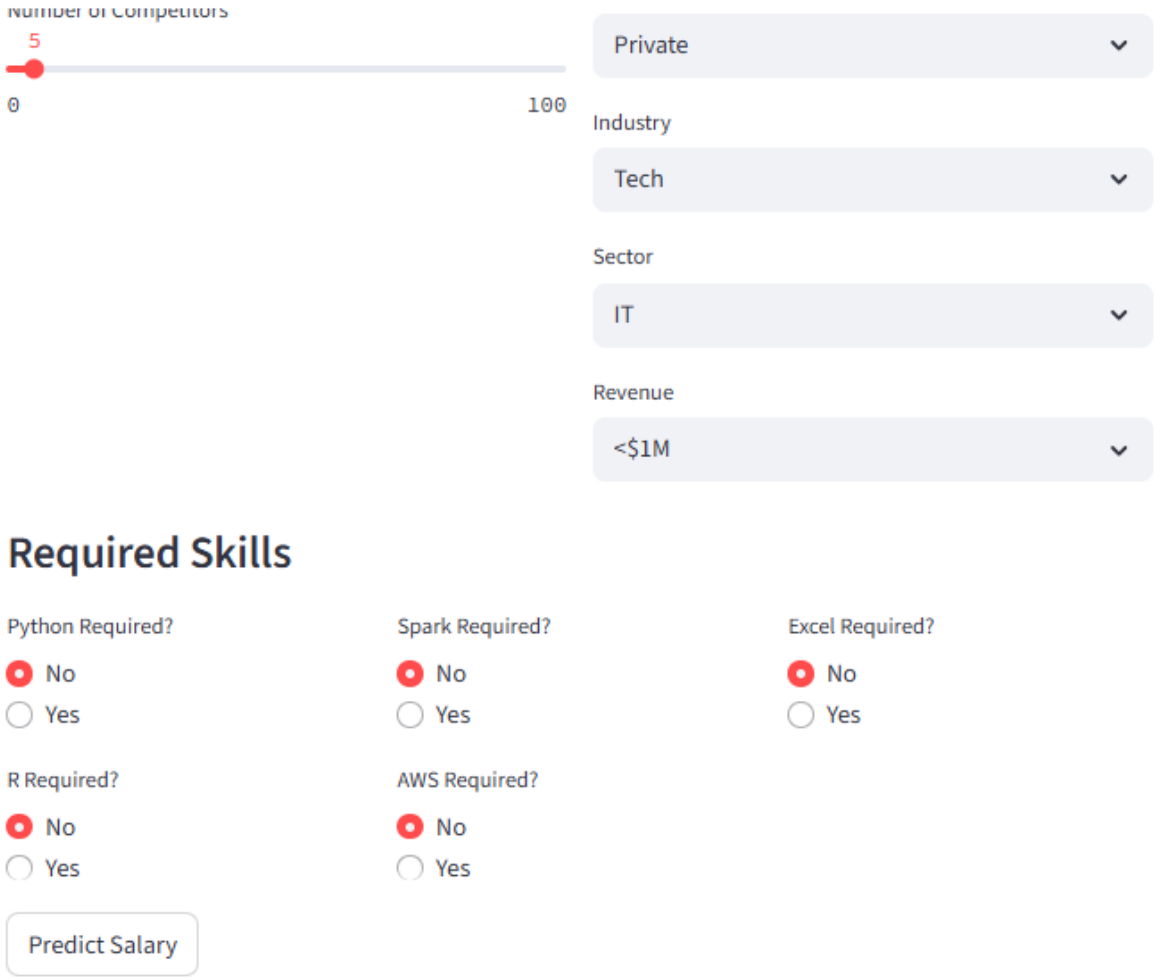
From fig we can see, many points lie close to the red line i.e., the model predicts well in those ranges. Some points deviate meaning model struggles with very high salary prediction. Overall, the random forest does a decent job, but there's some bias toward predicting around 100 since many dots is clustered horizontally there.

Saving the model

In [62]:
```
import joblib

joblib.dump(best_model, 'rfr_prediction_model.pkl')

joblib.dump(le, 'seniority_encoder.pkl')

training_columns = df.drop('avg_salary', axis=1).columns
joblib.dump(training_columns, 'training_columns.pkl')

joblib.dump(scaler, 'scaler.pkl')
```

Model Deployment





Conclusion:

In this project, I built a machine learning model to predict the average salary for a given jobs using job and company attributes. After exploring and analyzing the dataset, preprocessing steps such as encoding categorical variables and handling redundant columns were performed.

Several models including Random Forest Regressor, XGBoost, Linear Regression, K-Neares Neighbors, and Support Vector Regression were trained and compared. Among them, the Random Forest Regressor performed the best, achieving the highest R^2 score of 0.7810. Hyperparameter tuning further improved the models performance.

Finally, the model was deployed as a Streamlit web application, allowing users to input job details and receive salary prediction.